MANUAL
ON
VARIAN MINI-COMPUTER SYSTEM
UNDER VORTEX OPERATION

DEPARTMENT
of
CHEMICAL ENGINEERING

University of Queensland
MANUAL
ON
VARIAN MINI-COMPUTER SYSTEM
UNDER VORTEX OPERATION

(August 1975)

VORTEX

Varian Omnitask Real Time Executive

E.T. White & R.J. Marshall
6.8.75

Department of Chemical Engineering
University of Queensland
Brisbane, AUSTRALIA 4067

M. WILLIAMS
PREFACE

This volume is an instruction manual for the use of the Varian mini-computer system operating under VORTEX. It is in two parts. Part I describes the system and its physical operation. Part II covers details of operation under the VORTEX software package.

This manual was written as an introductory teaching manual. Material has been selected for ease of use and understanding. Greater detail on physical items or the VORTEX software is available in manuals provided by VARIAN. The beginner however, should not need to access these for initial applications.

VARIAN DOCUMENTATION

This Department has only a very limited number of the documents provided by VARIAN. Their absence or loss would be a very serious inconvenience to us so they must not be removed from the computer room.

Documents:

System: V72 Handbook
ADAPTS: ADAPTS Users Guide
ADAPTS Software Library Catalogue
Software: VARIAN Software Manual
individual software manuals
Hardware: each equipment item (e.g. cassettes, disc, teletype) usually has an instruction manual provided
Maintenance, Service: Considerable documentation

A copy of the VORTEX and ADAPTS manuals will be made available near the teletype. Other documents can only be obtained on request.

REQUEST FOR SUGGESTIONS ON IMPROVEMENTS

As the first attempt to produce a working manual for teaching purposes, it is very probable it has weaknesses and errors in various areas. We would very much welcome comments and suggestions for improvements for future versions.
PART II
OPERATION UNDER VORTEX

1. Introduction
   1-1 General Comments
   1-2 Presentation of Material

2. Description of VORTEX
   2-1 Modular Organisation
   2-2 Flexibility
   2-3 Functioning of the System
   2-4 Real Time Operation
   2-5 Input-Output Control

3. Use of Input/Output Control
   3-1 Need for Input/Output Connections
   3-2 Physical Devices
   3-3 Logical Units
   3-4 Commands for Input/Output Connections
   3-5 Default List of Logical Unit Connections

4. Use of VORTEX for Background Computation
   4-1 Introduction
   4-2 An Example
   4-3 Explanation of Operation
   4-4 Summary of Procedure Under VORTEX
   4-5 Other Examples
   4-6 Aborting the Running of a Program

5. FORTRAN IV Under VORTEX
   5-1 Languages Supported
   5-2 Comments on FORTRAN
   5-3 List of FORTRAN Statements
   5-4 BBASIC

6. Job Control Processor (JCP)
   6-1 Purpose
   6-2 Directives
   6-3 Scheduling JCP
   6-4 Notes on some JCP Directives

7. Source Edit (SEDIT)
   7-1 General Remarks
   7-2 General Organisation
   7-3 Logical Units
   7-4 Operation
   7-5 Comments on Examples
8. **File Maintenance (FMAIN)**

8-1 Introduction  
8-2 Logical Units  
8-3 FMAIN Directives

9. **Load Module Generation (LMGEN)**

9-1 Introduction  
9-2 Directives  
9-3 Logical Units  
9-4 Explanation  
9-5 Example

10. **Real Time Operation**

10-1 Introduction  
10-2 System Internal Interrupts  
10-3 External (Hardware Interrupts)  
10-4 Operator Communications (OC) Requests  
10-5 FORTRAN Statements Using RTE Facilities  
10-6 Control of Real Time Operation  
10-7 Priority Levels

11. **Entry and Recovery of Data From Files**

11-1 Introduction  
11-2 Directives  
11-3 Explanation of Operation  
11-4 File Control Block (FCB) Array  
11-5 Details of Commands  
11-6 Comments on Commands

12. **Data Logging and Control**

12-1 Hardware  
12-2 FORTRAN Statements for Input/Output Calls  
12-3 Control and Status Operation (PULSE,STAT)  
12-4 Organisation of VORTEX Process I/O Calls  
12-5 Calibration of Analog Input and Output  
12-6 Examples of FORTRAN Programs for Data Logging and Control

13. **Graphics**

13-1 Introduction  
13-2 List of Commands  
13-3 Scope of Graphics Software  
13-4 Hardware Organisation  
13-5 Software Organisation  
13-6 Details of Commands  
13-7 Examples
APPENDICES

1. Examples of Procedure for Running FORTRAN Programs
2. Summary of FORTRAN IV Commands Under VORTEX
3. Summary of Commands for JCP, SEDIT, FMAIN, LMGEN, OC
4. Library of FORTRAN Programs
5. General Procedure for Problem Solving Using the Digital Computer
6. Preparation of VARIAN Compatible Tape Using PDP 10

SUPPLEMENT

Selected Pages From VARIAN VORTEX Manual
PART I

DESCRIPTION OF

THE SYSTEM
PROCESSOR
V72
CENTRAL PROCESSING UNIT
CPU
(32K 16-bit core memory
660ns cycle time)

Printer
(LP)
360 lpm

Graphics display
360 characters/sec

Keyboard CRT (CT)

T1
36 ch/sec
(TV)

Video display
85.5 characters/sec

TY10 (VDU)

TY20 (in lab)
75 characters/sec

Paper tape punch
360 characters/sec

Paper tape reader

Dual Disc Unit
2.34 Mwords

Cassette Unit
120 Kwords/cassette

BULK STORAGE DEVICES

INTERFACE CONSOLE
16 analog in
4 analog out
32 digital in/out
32 status/control

analog digital
in & out in & out
to lab

INTERFACE DEVICES

INPUT-OUTPUT DEVICES

CONFIGURATION OF VARIAN COMPUTER SYSTEM
FIGURE 1-1  CONFIGURATION OF VARIAN COMPUTER SYSTEM.
General computation covers number crunching, record keeping, and display activities. Data Logging is when data from an external source is recorded automatically (logged) by the computer system. For example, the electrical output from a process instrument may be connected to the computer through the interface console, and the computer then used to sample the output and store the numbers within the computer.

If the computer is also used to generate signals, which then are transmitted to the process and used to control it, this is the third function, digital process control.

All three application areas will be discussed in Part II with operation under VORTIX.

1-4 Mode of Operation

The system may be operated under various packages of software. Of these, two are significant,

- VORTIX - a package that allows the use of FORTRAN and other languages and also allows multitasks (several users).
- ADAPTS - a package allowing the use only of FORTRAN. It is simple to use but is only single user.

Only one of these packages is available on the system at any time. At present emphasis is given to the VORTIX package, and this is discussed in detail in Part II of this manual.

1-5 Availability

Subject to demand and use, the departments' policy is to make the system available as extensively as possible to staff and students of this department.

It must be emphasised that this service can ONLY be made available to members of this department - that is, only to chemical engineering staff and students. This restriction must be strictly observed.

It is hoped that personnel of the department will use the system for the bulk of their computing requirements (prescribed and voluntary), and if usage allows, for recreation or idle enjoyment.
2. RULES AND GENERAL REMARKS

2-1 Limitation to Chemical Engineering Personnel

The system can only be used by students and staff of the Chemical Engineering Department. THIS IS TO BE STRICTLY OBSERVED.

2-2 Care and Responsibility

It is expected that all care will be taken with the equipment, and that sensible behaviour will prevail in its use.

For example do not use undue force on any equipment, do not skylark about near the machine. If a failure of any kind is observed report it immediately - do not try to press on.

We seek the cooperation and responsibility of all users. With this the greatest service and convenience to all will be achieved.

2-3 System Failure

Report all cases of system failure, malfunction or confusion to the operators via the FAILURES book. Give the fullest details including relevant print out. If the system crashes outside of 9 to 5 and you are at loss what to do contact one of the emergency phone numbers listed on the wall.

2-4 Booking Rules

The system will be available 24 hours a day, 7 days a week. To give fair and reasonable access to all, certain rules limiting use in prime time and allowing orderly booking for future use have been introduced. These are not aimed to restrict use, but solely to give fair allocation to all users.

2-5 Program Storage

Programs may be stored on disc. Users should also arrange to obtain a separate personal copy on paper tape or cassette, in case the contents of a disc are accidently destroyed.

Programs stored on disc must be prefixed by two characters identifying the programmer. For undergraduate students these characters are, ...

x2 - second year students
x3 - third year students
x4 - fourth year students

Graduate students and staff have separate codes as notified elsewhere.
Disc storage eventually is limited. Periodically discs will be cleared of accumulated programs. Those without identifying initial characters will be deleted at once. Those identified will be checked with the user before a decision on retaining or deleting is made.

2-6 Program Library

It is intended that as programs are developed and documented, these will be kept as a library of programs for future users. Periodically a list of library programs available will be prepared, and a copy will be available from the department.

To use a library program, it will be necessary to request the operator to load it onto disc for your later use.

2-7 Cost of System

The purchase cost of the system was about $55,000.

Individual items, (approx.)

- C.P.U. + core, cabinet and power supply $15,000
- Disc Unit $11,000
- Cassette Unit 3,000
- Interface Console 5,000
- Teletype + cassette 4,000
- C.R.T. terminal 4,000
- printer 6,000
- Paper tape punch and reader 3,000
- X - Y plotter 1,000

Cost of spare discs $150, digital cassettes $5 to $10. Paper for TI teletype $6 per roll

2-8 Cassettes

Digital cassettes may be hired from the department on payment of a deposit. The deposit is repaid on return of the cassette in good condition.
3. CENTRAL PROCESSING UNIT AND CORE MEMORY

3-1 The Varian Minicomputer System

As described in Section 1, the system can be considered in four parts.

(i) the central processing unit and core memory.
(ii) the disc and cassette bulk storage devices.
(iii) the input/output devices
(iv) the interface console

These are shown in figure 1-1, which indicates the complement of input/output devices presently available.

These parts will be described further. Sufficient detail is given to cover normal usage. Extra details for the advanced user are given in the appropriate manuals.

3-2 The Central Processing Unit (CPU) and Core Memory

The system is built around a Varian V72 central processing unit, with 32K of 16 bit core memory (cycle time 660 microseconds), and three working registers (A,B,X).

The front panel contains a key switch, a number of displays, and a variety of buttons.

Under normal operation it will not be necessary to use any of these. The only display of concern is the RUN light. This will be alight when the system is operational. If it blinks or goes out (the STFP button may then alight) the system has 'crashed' (become inoperative). In this case, help should be sought.

The key switch should normally be locked in the CONSOLE DISABLE position; in which case pressing the console buttons will not cause any action.

3-3 Available Memory

Although the computer has 32K of core, machine tasks and the VORTEX software nucleus takes a substantial portion (approximately 14K). This amount could change as further software packages are added. The residual memory is allocated between the various tasks running at the one time.

Nevertheless the computer can handle a fairly large program, for example one of many hundred FORTRAN statements. The computer will indicate with an error message if there is not sufficient core available. If a large program requires more core than is available, it is often possible to split it into portions which are stored on disc, and these portions (overlays) can be brought in as required.

* 1K = 1024
If the system is being used to store data values, with careful operation, 7 or 8K of values may be stored in core. If practical, of course, it would be better if these could be transferred to a bulk storage device (see Section 4) which gives a much higher capacity.

3-4 Run Time

It is often useful to be able to make an estimate of the time it will take and program to run, particularly when a portion of a program is repeated a large number of times. The following are very approximate estimates of some of the basic operations in the machine.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition, substraction</td>
<td>0.4 ms</td>
</tr>
<tr>
<td>Multiplication, division</td>
<td>0.5 ms</td>
</tr>
<tr>
<td>Raise to a power</td>
<td>4 ms</td>
</tr>
<tr>
<td>Log, exp, sine etc.</td>
<td>2 ms</td>
</tr>
<tr>
<td>Decision (simple IF)</td>
<td>1 ms</td>
</tr>
</tbody>
</table>

An estimate of the run time can be obtained by summing the various operations in a loop and multiplying by the number of repetitions (iterations) of that loop.

For example, if the calculation $y = a\cdot x^b + c$ is to be calculated for 1000 values of $x$, the time taken will be about 5 seconds.
4 Bulk Storage Devices

4-1 Introduction

Although there are 32K words of core memory available in the processor, this is normally only sufficient to hold running programs and calculated values.

Programs not being run may be kept in a bulk storage device. The bulk storage device can also be used to store data, or portions of large programs which are too big for the core memory available.

There are two bulk storage devices on the system

(a) the disc unit
(b) the cassette storage unit

Of these the disc unit is the most important.

4-2 Disc Unit

A 'Diablo' twin platter moving head-disc unit is installed. One platter is fixed (cannot be removed), while the other is removable. Each platter can store about 1.2 million words, and data can be transferred at rates of 1.5 M baud (million bits per second) i.e. at about 90,000 words per second.

Each disc of magnetic material is rotated at a high speed (1500 rpm), while the heads can move radially to locate the section of the disc on which to store or read information magnetically.

The disc controls consist of a keyed switch and six buttons (five push, one rocker). Under normal operation it will not be necessary to access these controls. The keyed switch prevents the disc unit from being removed, or from being restarted if the unit 'crashes'. When the unit is operational the POWER button and the READY button should be alight. If these are not, seek help.

Transfer of information to and from the disc is achieved by suitable program commands (refer to Part II of manual).

4-3 Cassette Unit

The cassette unit allows users of the system to have their own copy of programs, data, etc., which can be fed immediately into the computer.

# Several measures of storage capacity are in general use; the bit (binary digit), the byte (8 bits) and the word (in the case of this computer of 16 bits). For this machine a character (e.g. A) is stored in 1 byte (1/2 word) while a number (e.g. 7,7) is stored as 4 bytes (2 words).
The cassette unit is similar in principle to cassette recorder except that it records digital information onto special digital grade magnetic tape. Special formatted tape cassettes are required. Each tape can store about 120 K words of information; at a transfer rate of about 0.4 K words/sec. It can take up to a minute to get information from a cassette, depending on the amount of information and its location on the tape.

This cassette unit is not normally available to undergraduate students, but they may use it on supervisors' advice for thesis or project work. Staff and graduates students are encouraged to use this means of personal program storage. Cassettes may be hired from the department.

Operating Instructions for Cassette Unit

There are only two controls on the cassette unit, the POWER and the MOTION buttons.

**IMPORTANT**

(a) The MOTION button must always be out when cassettes are inserted or removed. This button raises and lowers the reading heads.

(b) Both channels of the cassette tape are read together: one contains addresses, the other program or data. Thus the cassette must always be inserted correctly i.e. with the printed side outwards.

(1) Ensure POWER is on; button lights up on depressing.

(2) Depress MOTION button till out. (IMPORTANT)

(3) Open door and insert cassette firmly (tape up, correct side out).

(4) Close door, depress MOTION button. (it flashes).

(5) The Cassette may now be controlled from program instructions or commands. In operation the MOTION light has three states

(i) on - tape is moving.
(ii) flashing - tape heads not reading or writing tape.
(iii) off - head on tape (OR power off).

(6) To remove cassette, press MOTION button till out, open door, remove cassette. Close door, switch off POWER.

Transfer of information to and from the cassette unit is actuated by appropriate software commands (refer Part II).
PART II

OPERATION

UNDER

VORTEX
1. INTRODUCTION

1-1 General Comments

VORTEX = Varian Omnitask Real Time Executive

The VORTEX software is a very versatile and powerful package. It allows the computer to perform several tasks at once (omnitask).

The principal tasks are those concerned with real time operation. These are carried out in what is termed the foreground portion of the computer memory. Real time processing allows,

* tasks to be scheduled at prescribed real times;
* tasks to be interrupted for more urgent ones;
* low priority tasks to be carried out when the computer is not otherwise busy.

This facility of real time operation is vital for any computer which is to be used for plant control purposes.

There is no prescribed limit to the number of programmes which can be running in the foreground. Practical limitations are the availability of computer time to perform the functions in each programme, and the availability of sufficient memory to store them.

Under the VORTEX package, the foreground can be used only for the RUNNING of programmes. In other words, programmes can not be COMPILED, DEBUGGED or EDITED in the foreground. These are not "real time" needs.

The lowest priority portion of the computer memory is termed the background. This portion can be used for programme development (e.g. COMPILING, EDITING, DEBUGGING). Only one programme can be handled in the background at any time.

Thus under VORTEX, the computer can be running any number of programmes in the foreground (e.g. control operations, data logging, or number crunching), but ONLY one programme (development or running), can be operational in the background.

The background task is really a real time operation operating under a software package called the job control processor. However, because this background task is allocated the lowest priority within the computer, it is a "time fill in" operation and thus will not be considered as a real time operation in the above sense.

1-2 Presentation of Material

Although the real time operation is the key to the system operation, newcomers to the system probably first will gain experience (or build on previous experience) by using the computer in the non-real time sense, i.e. for programme development and running in the background.
The material in this part of the manual will therefore be developed as follows:

(i) description of VORTEX (Section 2);

(ii) use of VORTEX for background computation (Section 4);

(iii) real time operations under VORTEX (Section 10);

(iv) special remarks concerning control and data logging uses (Section 11).

The VORTEX package supports a number of software languages. Of these, FORTRAN is the most important for our needs and its use will be discussed predominantly in this part of the manual.

Other languages (e.g. RPGIV) are described in the Varian software manual.
2-1 **Modular Organisation**

The VORTEX package comprises a number of software modules each with its own task. Each software module has its own directives (commands) and error messages.

For example one obvious task is to 'oversee' and control the computer operations and there is a software module (the REAL TIME EXECUTIVE) which does this. Another task is to control programme development in the 'background' memory (the JOB CONTROL PROCESSOR module), while another is the FORTRAN compiler. A full list of these modules is given in Table 2-1.

2-2 **Flexibility**

With VORTEX the user is given a system of considerable power and versatility. Versatility naturally means a larger choice of options, and therefore a larger number of decisions to be made by the user. Hence with VORTEX it takes a little longer to organise a job, since a number of decisions have to be made initially (which software module to use? On which device are the results to be displayed? etc.) To make these decisions, some knowledge of the functioning of the computer is necessary.

2-3 **Functioning of the System**

The power of the VORTEX package is associated with two main aspects,

(i) real time operation;
(ii) individual selection of input and output devices.

These will be discussed in turn.

2-4 **Real Time Operation**

As mentioned (Section 1), real time operation requires the system to respond immediately to an urgent need.

This is organised through PRIORITY INTERRUPTS. In effect, the system scans a set of locations to see if an interrupt (a signal change) has occurred. If an interrupt has occurred its priority is checked. If it refers to a programme with priority higher than the programme already being executed, the present programme will be halted,
<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
<th>Fore-or Back-Ground</th>
<th>Error Message Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL TIME EXECUTIVE</td>
<td>control whole system</td>
<td>Both</td>
<td>EX</td>
</tr>
<tr>
<td>OPERATOR COMMUNICATIONS</td>
<td>operator interrupt of system operation.</td>
<td>F</td>
<td>OC</td>
</tr>
<tr>
<td>INPUT/OUTPUT CONTROL</td>
<td>interconnect I/O devices</td>
<td>F</td>
<td>IO</td>
</tr>
<tr>
<td>JOB CONTROL PROCESSOR</td>
<td>background operation control</td>
<td>F</td>
<td>JC</td>
</tr>
<tr>
<td>FORT (Fortran IV Compiler)</td>
<td>compiles FORTRAN programs</td>
<td>B</td>
<td>special</td>
</tr>
<tr>
<td>BBASIC (Basic)</td>
<td>handles BASIC programs</td>
<td>B</td>
<td>special</td>
</tr>
<tr>
<td>DASMR (Assembler)</td>
<td>compiles Assembler programs</td>
<td>B</td>
<td>special</td>
</tr>
<tr>
<td>CONC (Concordance)</td>
<td>indexed listing of programs</td>
<td>B</td>
<td>CM</td>
</tr>
<tr>
<td>RPG (report Program Gen.)</td>
<td>report generation software.</td>
<td>B</td>
<td>RP, RT</td>
</tr>
<tr>
<td>SEDIT (Source Editor)</td>
<td>editing source programs</td>
<td>B</td>
<td>SE</td>
</tr>
<tr>
<td>FMAIN (File Maintenance)</td>
<td>maintaining files</td>
<td>B</td>
<td>FM</td>
</tr>
<tr>
<td>SMMAIN (System Maintenance)</td>
<td>maintains library</td>
<td>B</td>
<td>SM</td>
</tr>
<tr>
<td>LMGGEN (Load Module Generator)</td>
<td>transfer to foreground</td>
<td>B</td>
<td>LG</td>
</tr>
<tr>
<td>DEBUG (Debugging)</td>
<td>debugging</td>
<td>B</td>
<td>DG</td>
</tr>
<tr>
<td>SNAP (Snapshot)</td>
<td>snapshot dump</td>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** Some other software modules are listed in the VORTEX manual (e.g. system generation, dataplot).

**Table 2-1:** Listing of Some of the Software Modules Available Under VORTEX
and the higher priority one executed. After completion the programme with the next highest priority will be executed. The disc unit is available for storage of background programmes temporarily squeezed out by foreground programmes.

The system incorporates a REAL TIME CLOCK (a quartz crystal controlled oscillator). Using this it is possible to schedule operations to be carried out at certain times (e.g. 11am) or after specified time intervals. These requests are treated as priority interrupts and have their own priority levels.

It is also possible to interrupt operations by using one of the teletypes (TTY2) which is the OPERATOR COMMUNICATION (OC) device. This allows the operator to type in commands to directly control the computer operations. This set of commands is another module of the VORTEX package (see Table 2-1) and are discussed later (Section 10). It is only possible to operate these message commands on the OC teletype and not on the other peripherals. Thus this facility will not normally be available for general student use.

There are 32 priority levels (0 through 31) available. The lowest levels 0 and 1 are for BACKGROUND tasks, while the higher priority levels (2 through 31) are for tasks in the FOREGROUND. Only the 'background' can be used for programme development and this is organised under the JOB CONTROL PROCESSOR module (a priority 1 task). The highest priority levels are kept for vital operations such as protecting the memory and saving the contents of the computer in case of power failure (POWER FAIL/RESTART).

The control of the real time operations of the machine is through the REAL TIME EXECUTIVE software module. It allows the scheduling, starting, suspending, delaying, aborting, etc., of tasks. These can be FORTRAN commands if desired. These will be considered further under real time operation (Section 10).

Operation in the 'background' is under control of the JOB CONTROL PROCESSOR module. This can call up other modules such as the FORTRAN compiler, the source editor (SEDIT), the file maintenance (FMAIN) software and the LOAD MODULE GENERATOR (LMGEN). The LMGEN software, for example, can be used to create "load modules" for programmes to be run in the foreground.

2-5 Input-Output Control

The second important aspect of VORTEX operation is the choice of input/output devices. For example, for some situations the output may be wanted on the teletype, for others on the printer (because it is a faster device) and for others on the visual display unit (because it is still faster and no permanent record of the output is required). It is left to the user to make the choice. How this is done is described in the next section.
3. USE OF INPUT/OUTPUT CONTROL

3-1 Need for Input/Output Connections

The computer system has a number of input and output peripheral devices (e.g. various teletypes, printer, cathode ray screen, paper tape units) on which information can be sent or received. It also uses various portions (PARTITIONS) of the disc storage unit for programme storage or programme work areas. All these (actual physical input/output devices and areas on the disc) will be termed PHYSICAL DEVICES.

The software packages are written as software modules. To retain high flexibility they are written in terms of various LOGICAL UNITS for input/output and storage. For example all information coming out as listing as a result of some of the software will be sent to a logical unit labelled LO (list output). All these logical units are also given LOGICAL UNIT NUMBERS (e.g. LO = 005), and can be referenced by either name or number.

Thus to effectively run the system it is necessary to connect the necessary logical units required in the software module to the appropriate physical devices.

For example under the JOB CONTROL PROCESSOR the directive

/ASSIGN, LO = CT00

will connect LO the list output to the cathode tube display (CT00). Any information coming out at the LO logical unit as the programme runs will therefore be displayed on the cathode tube.

To ensure that there is always some connection to each logical unit, a DEFAULT list of connections is drawn up at system generation. (System generation is carried out at very occasional intervals to reconfigure the system for optimum use.) If connections are not assigned by the user, the DEFAULT connections are used.

3-2 Physical Devices

The physical devices available on our system are given in Table 3-1. Each has a code name as indicated.

Note that some physical devices are visible input/output devices while others are areas on the disc storage. The code names for disc storage (and the space available) can be altered at system generation. (This facility is not available for the general user.)

The dummy device is a means of quickly disposing of output which is not wanted. It is a software "physical device" and can also be described as a "logical unit" with a "logical unit number" (see next Section).
Table 3-1  List of Physical Devices

<table>
<thead>
<tr>
<th>Device</th>
<th>Code Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teletype Ø (TTY Ø)</td>
<td>TØØ or TY</td>
</tr>
<tr>
<td>Teletype 1 (TTY 1)</td>
<td>TØ1</td>
</tr>
<tr>
<td>Teletype 2 (TTY 2)</td>
<td>TØ2</td>
</tr>
<tr>
<td>Cathode Tube (KBCRT)</td>
<td>CTØØ or CT</td>
</tr>
<tr>
<td>Paper tape punch and reader</td>
<td>PTØØ or PT</td>
</tr>
<tr>
<td>Line Printer</td>
<td>LPØØ or LP</td>
</tr>
<tr>
<td>Cassette</td>
<td>---</td>
</tr>
</tbody>
</table>
| Disc areas                          | DØ0A, DØ0B, DØ0C, etc.
|                                     | DØ1A, DØ1B, etc.     |
| Dummy (no device, a hypothetical dump) | DUM (Logical unit ØØØ, or Ø) |

3-3 Logical Units

Inputs and outputs from the software are directed to a LOGICAL UNIT. Table 3-2 lists the major logical units involved in VORTEX. Each logical unit has a 'logical unit number' (lun). When connections are made to physical devices, the logical unit may be referenced either by its code (e.g. L0) or by its number (Ø05). In many (but not all) cases these references are interchangeable.

Other logical units (up to a total of 256) can be prescribed at system generation for special purposes.

Of those listed, the processor input (PI) is the logical unit for the inputting of source statements (e.g. of FORTRAN statements) L0 the output for the listing of a programme and PØ the output of the calculations. SI and S0 are input and output for directives (commands) and messages to the system. For example an instruction to compile a FORTRAN programme (/FORT) enters at logical unit SI. Binary input and output are routed through BI and BO and so on.

Table 3-3 lists the logical units involved in a number of the software models. Greater detail is given in the appropriate later section. For example, when the source editor (SEDIT) is used the logical units indicated will be used. The user should ensure that these are connected to the physical devices he requires.
# TABLE 3-2 LIST OF MAJOR LOGICAL UNITS

<table>
<thead>
<tr>
<th>Logical Unit No.</th>
<th>Name</th>
<th>Description</th>
<th>Function</th>
<th>Allowable physical devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DUM</td>
<td>Dummy</td>
<td>For I/O simulation</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>OC</td>
<td>Operator communication</td>
<td>For system operator communication with immediate return to user control</td>
<td>TTY, CT only</td>
</tr>
<tr>
<td>2</td>
<td>SI</td>
<td>System input</td>
<td>For inputs of all JCP control directives to any device.</td>
<td>Anywhere</td>
</tr>
<tr>
<td>3</td>
<td>SO</td>
<td>System output</td>
<td>For display of all input control directives and output system messages</td>
<td>TTY, CT only</td>
</tr>
<tr>
<td>4</td>
<td>PI</td>
<td>Processor input</td>
<td>For input of source statements from all operating system language processors.</td>
<td>Anywhere</td>
</tr>
<tr>
<td>5</td>
<td>LO</td>
<td>List output</td>
<td>For output of operating system input control directives, system operations messages, and operating system language, processors' output listings.</td>
<td>Anywhere (+ LP)</td>
</tr>
<tr>
<td>6</td>
<td>BI</td>
<td>Binary input</td>
<td>For input of object-module records from operating system processors.</td>
<td>Disc, paper tape, Dum only</td>
</tr>
<tr>
<td>7</td>
<td>BO</td>
<td>Binary output</td>
<td>For output of object-module records from operating system language processors.</td>
<td>Ditto</td>
</tr>
<tr>
<td>8</td>
<td>SS</td>
<td>System scratch</td>
<td>For system scratch use; all operating system language processors that use an intermediate scratch unit input from this unit.</td>
<td>Disc, Dum</td>
</tr>
<tr>
<td>9</td>
<td>GO</td>
<td>Go unit</td>
<td>For output of the same information as the BO unit by the system assembler and compiler.</td>
<td>Disc</td>
</tr>
<tr>
<td>10</td>
<td>PO</td>
<td>Processor output</td>
<td>For processor output, all operating system language processors that use an intermediate scratch unit output to this unit.</td>
<td>Same device as SS</td>
</tr>
<tr>
<td>11</td>
<td>DI</td>
<td>Debugging input</td>
<td>For all debugging inputs</td>
<td>TTY, CT only</td>
</tr>
<tr>
<td>12</td>
<td>DO</td>
<td>Debugging output</td>
<td>For all debugging outputs</td>
<td>TTY, CT, LP</td>
</tr>
<tr>
<td>101</td>
<td>CU</td>
<td>Checkpoint unit</td>
<td>For use by VORTEX to checkpoint a background task; partition protection key S.</td>
<td>Disc</td>
</tr>
<tr>
<td>102</td>
<td>SW</td>
<td>System work</td>
<td>For generation of a load module by the system Disc load module generator component; or for cataloging loading or execution by other system components; partition protection key B.</td>
<td>Disc</td>
</tr>
<tr>
<td>103</td>
<td>CL</td>
<td>&quot;Core&quot; resident library</td>
<td>For all &quot;core&quot; resident system entry points; partition protection key C.</td>
<td>Disc</td>
</tr>
<tr>
<td>104</td>
<td>OH</td>
<td>Object-module library</td>
<td>For the VORTEX system object module library; partition protection key D.</td>
<td>Disc</td>
</tr>
<tr>
<td>105</td>
<td>BL</td>
<td>Background Library</td>
<td>For the VORTEX system background library; partition protection key E</td>
<td>Disc</td>
</tr>
<tr>
<td>106</td>
<td>FL</td>
<td>Foreground Library</td>
<td>For the VORTEX system foreground library; partition protection key F.</td>
<td>Disc</td>
</tr>
</tbody>
</table>

Other logical units can be assigned as user foreground libraries provided they are specified at system-generation time. However, there is only one background library in any case.
Table 3-3: Logical Units Involved in Some of the Software Modules

<table>
<thead>
<tr>
<th>Software Module</th>
<th>Logical Units Accessed</th>
</tr>
</thead>
<tbody>
<tr>
<td>JCP</td>
<td>SI (for inputting directives); all other logical units through/ASSIGN</td>
</tr>
<tr>
<td>SEDIT</td>
<td>SI (input SEDIT directives), LO (listing), SO (error processing) IN, OUT, ALT through AS directives (PI, PO, BI only as default for IN, OUT, ALT).</td>
</tr>
<tr>
<td>FORT</td>
<td>PI (source records input)</td>
</tr>
<tr>
<td></td>
<td>BO (binary &quot;object module&quot; (compiled program))</td>
</tr>
<tr>
<td></td>
<td>LO (listings)</td>
</tr>
<tr>
<td>BBASIC</td>
<td>#/ (input statements), 21 (output statements and error messages), 22 (run/interactive and error messages), PO (punch statements), PI (alternate input statements).</td>
</tr>
<tr>
<td>FMAIN</td>
<td>SI (input LGEN directives)</td>
</tr>
<tr>
<td></td>
<td>LO (listings), SO (error processing)</td>
</tr>
<tr>
<td></td>
<td>SW (temporary work unit)</td>
</tr>
</tbody>
</table>

3-4 Command for Input/Output Connections

The command for connecting physical devices to logical units is the /ASSIGN directive as described above. It is one of the JOB CONTROL PROCESSOR directives and will be considered again in Section 6.

Some examples of its use.

/ASSIGN, PI = TY00, PO = TY, LO = CT.

or

/ASSIGN, 4 = TY, 10 = TY, 5 = CT00

(If a physical device has 00 as the last two characters, they may be omitted if desired.)

If any logical unit is not reassigned it will be at the default assignment. Obviously if the default arrangement is suitable, there is no point in assigning it again.

At the end of each job the assignments revert to the default values.

3-5 Default list of Logical Unit Connections

A typical list of default connections for the logical units is given in Table 3-4.

At system generation, alterations may be made. A current list will be displayed on the notice board in the computer room.
### Table 3-4

**LIST OF DEFAULT LOGICAL UNIT ASSIGNMENTS (ÖILIST)**

<table>
<thead>
<tr>
<th>Logical Unit</th>
<th>Lun</th>
<th>Physical Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>OC</td>
<td>001</td>
<td>TY10</td>
</tr>
<tr>
<td>SI</td>
<td>002</td>
<td>TY00</td>
</tr>
<tr>
<td>SO</td>
<td>003</td>
<td>TY00</td>
</tr>
<tr>
<td>PI</td>
<td>004</td>
<td>TY00</td>
</tr>
<tr>
<td>LO</td>
<td>075</td>
<td>DUM</td>
</tr>
<tr>
<td>BI</td>
<td></td>
<td>D001</td>
</tr>
<tr>
<td>BO</td>
<td>007</td>
<td>D001</td>
</tr>
<tr>
<td>SS</td>
<td>008</td>
<td>D00H</td>
</tr>
<tr>
<td>GO</td>
<td>009</td>
<td>D00G</td>
</tr>
<tr>
<td>PO</td>
<td>010</td>
<td>D00H</td>
</tr>
<tr>
<td>DI</td>
<td>011</td>
<td>CT00</td>
</tr>
<tr>
<td>DO</td>
<td>012</td>
<td>CT00</td>
</tr>
<tr>
<td>CU</td>
<td>101</td>
<td>D00E</td>
</tr>
<tr>
<td>SW</td>
<td>102</td>
<td>D00F</td>
</tr>
<tr>
<td>CL</td>
<td>103</td>
<td>D00A</td>
</tr>
<tr>
<td>OM</td>
<td>104</td>
<td>D00D</td>
</tr>
<tr>
<td>BL</td>
<td>105</td>
<td>D00C</td>
</tr>
<tr>
<td>FL</td>
<td>106</td>
<td>D00B</td>
</tr>
<tr>
<td>IT</td>
<td>013</td>
<td>TY00</td>
</tr>
<tr>
<td>OT</td>
<td>014</td>
<td>TY00</td>
</tr>
<tr>
<td>X2</td>
<td>015</td>
<td>D00K</td>
</tr>
<tr>
<td>X1</td>
<td>016</td>
<td>D00J</td>
</tr>
<tr>
<td>MS</td>
<td>017</td>
<td>D01A</td>
</tr>
<tr>
<td>PC</td>
<td>018</td>
<td>D01B</td>
</tr>
<tr>
<td>DU</td>
<td>023</td>
<td>DUM</td>
</tr>
<tr>
<td>PT</td>
<td>024</td>
<td>PT00</td>
</tr>
<tr>
<td>LP</td>
<td>025</td>
<td>LP00</td>
</tr>
<tr>
<td>AT</td>
<td>026</td>
<td>TY10</td>
</tr>
<tr>
<td>BT</td>
<td>027</td>
<td>TY20</td>
</tr>
<tr>
<td>CT</td>
<td>028</td>
<td>CT00</td>
</tr>
<tr>
<td>II</td>
<td>207</td>
<td>CI00</td>
</tr>
<tr>
<td>OI</td>
<td>201</td>
<td>CO00</td>
</tr>
<tr>
<td>O2</td>
<td>202</td>
<td>CO10</td>
</tr>
</tbody>
</table>

*New for our system*
4-1 USE OF VORTEX FOR BACKGROUND COMPUTATION

4-1 Introduction

The problem of running a programme in the background is probably the best introduction to VORTEX. With this experience, the extension to other VORTEX facilities should be easier.

To demonstrate the use of VORTEX for background computation, consider a simple example.

4-2 An Example

The example illustrates the entering of a FORTRAN programme under VORTEX by typing in through a teletype. The programme ends up stored in a scratch area called USER on the disc. A copy of the teletype print out is, -

```
/JOB, JOB1
JC**
/ASSIGN, LO=CT00
JC**
/SEDIT
SE**
AS, IN=PI
SE**
AS, OU=15,, USER
FC
C    TEST PROGRAM
A=10.0
B=5.0
C=1.0
D=A+B+C
WRITE (14,10)A,B,C,D
10 FORMAT (4G15.7)
STOP
END

/ENDJOB
JC**
```

Job Control Processor (JCP) commands

Source edit (SEDIT) commands

FORTRAN statements

ended with EOF (CTRL BEL)

4-3 Explanation of Operations

(i) Analysis of requirements

The programme is to be entered under the SEDIT software module. This is called up under the JCP software module (Table 2.1). Thus two software modules are involved. The overall procedure then will be, -

(i) call up JCP,
(ii) make all necessary assignments of logical units to physical devices,
(iii) call up SEDIT,
(iv) make any necessary assignments under SEDIT,
(v) insert the given programme,
(vi) call back JCP to end job.
Now we have to decide what assignments of logical units to physical devices are necessary. From Table 3.3 (or in more detail Sections 6 and 7) we see that JCP involves the logical unit SI and that SEDIT involves the logical units SI, LO and SO, while within SEDIT there are its own special logical units IN, OUT, ALT. Table 3.3 also shows that by default IN, OUT and ALT are assigned to the general logical units PI, PO, BI respectively. The default assignments of the general logical units SI, LO, SO, PI, PO and BI are given in Table 3-4. This means that the physical devices are connected to the logical units involved as follows:

TYØØ;SI,SO,PI(& IN),
DUM;LO
disc area DØØ;PO(& OUT)
disc area DØØ;BI(& ALT)

Are these suitable?

What do we want? To input JCP and SEDIT directives (on SI) we would like to use the teletype, while correcting SEDIT errors (SO) again we would like the teletype. The list output (LO) can conveniently go to the CRT. We would like the given programme to be typed in (IN) on the teletype, while the OUT should be connected to an unprotected file called USER on disc area DØØK (Ø15). We do not need the alternative input (ALT) at this stage, so we will not concern ourselves with it.

Thus all default assignments are suitable except for OUT, so we will have to assign this. To demonstrate the directives we will also assign IN to PI. This of course being the default assignment is a redundant directive.

So the procedure will now be:

(i) call up JCP,
(ii) assign LO to CTØØ,
(iii) call up SEDIT,
(iv) assign IN to PI, OUT to the USER file on file area X2 (Ø15) on DØØK,
(v) type in the programme,
(vi) call back JCP to end job.

(ii) Performing the operations

First ensure the system is operational and that the VORTEX package is loaded.

The JCP module should be available initially at the terminal. Type CTRL C <cr> (see Section 4-5) to abort any other programme that might be running in the background. Type /C <cr>; the system should respond with JCP indicating that the JCP software model is active. If there is no response seek help.

Now follow the example in Section 4-2.
All JCP commands begin with a slash / . After each command is executed under JCP, JCP** is printed to indicate to the user to continue. In the example, under JCP the /JOB initiates the job and names it JOB1 (any name will do). Then the various logical unit assignments are made. In this case the list output LO is transferred to the CRT, CTIP by the command/ASSIGN, LO=CTIP. CTIP can be shortened to CT.

The programme has to be entered under the SOURCE EDITOR (SEDEIT) software module, so the JCP directive (/SEDEIT) is used to call up this software module. SE** is printed to show SEDEIT is ready.

Under the SEDEIT software module, the input IN and output OU of the editing programme are connected to appropriate logical units. In this case IN is taken from PI which is connected to the TTY (Table 3-4) and the OU goes to logical unit 15 (X2) which is connected to the disc area D00K (Table 3-4). There it will be placed in a unprotected file (already present) called USER. See Section 7 for further details of this directive. The SEDEIT command FC (file copy) causes the statements following to be copied directly into the file USER on disc. Information is copied line by line (<cr> terminates a line), and lines will be copied until an EOF, end of file (CTRL BEL keys) character is transmitted.

The statements entering are FORTRAN statements, though statements in any language can be handled in this way.

After the EOF character, SEDEIT indicates by SE** it has completed the FC command. Typing a directive starting with a slash / transfers control back to the first software module (JCP). In this case /ENDJOB is used to indicate the job is finished. Note that at the /ENDJOB command all logical unit assignments revert to the default values. For the next job they will have to be reassigned again. If there were other work to be done, this could have been done before the /ENDJOB.

4-4 Summary of Procedure under VORTEX

The procedure under VORTEX is as follows, -

(i) analyse the requirements - what software modules?
   What assignments will be necessary?
(ii) call up JCP and make assignments,
(iii) carry out job,
(iv) change assignments if necessary,
(v) carry out next job, etc.,
(vi) finalise by /ENDJOB.

4-5 Other Examples

Other examples copying from paper tape, listing, editing, storing, compiling and executing FORTRAN programmes are given in Appendix I. In principle, they follow the basic organisation outlined above, although different logical unit assignments and other software modules have been used.
To carry out these operations under VORTEX, some knowledge is required of FORTRAN (Section 5), the JOB CONTROL PROCESSOR module (Section 6), the SEDIT module (Section 7) and the FILE MAINTENANCE module (Section 8).

### 4-6 Aborting the Running of a Programme

While a programme is being executed it may be stopped by typing CTRL C. This is organised as an 'unsolicited request' and will abort any present task and return control to JCP. If the machine is soliciting information e.g. under SEDIT or FMAIN this CTRL C will be ignored. In these cases a slash / will bring control back to the JCP.

If the computer gets hung up on a programme, it may be in a loop, but possibly it is a result of incorrect assignments. Use CTRL C to return control to JCP, and make assignments again to be sure. /ENDJOB will change all assignments to default values.
5. FORTRAN IV UNDER VORTEX

5-1 Languages Supported

VORTEX supports four language processors: assembler (DASHR), background BASIC (BBASIC), FORTRAN IV compiler and a report generation programme (RPGIV), plus the auxiliary CONCORDANCE programme (CONC).

Of these, FORTRAN is of present interest.

5-2 Comments on FORTRAN

The FORTRAN IV compiler is scheduled by the job control processor (JCP) directive /FORT (Section 6) and p.sq of the supplement. Note the options available for suppressing symbol table listing (M), source listing (N), etc.

The FORTRAN IV language processor available under VORTEX supports a wide range of FORTRAN statements. These are listed in Appendix 2.

It does not support all the FORTRAN facilities which are available on the PDP/10. The following differences from PDP/10 FORTRAN should be noted:

(i) file structure commands are different

(ii) subscripts in arrays may only start from 1 (not 0) i.e. the first term in the two dimensional array A is A(1,1)

(iii) the ability to handle mixed mode expressions is limited. Avoid involved expressions containing integer and real variables.

(iv) real numbers must have a decimal point i.e. 10. not 10 as a real number

(v) IMPLICIT type statements cannot be used

(vi) \$I for horizontal tab is not available

(vii) paper tapes generated on line by the PDP/10 are not directly compatible, although a programme is available on the PDP/10 to give a compatible tape. Details are given in Volume 1 of this manual. It is possible to use the Varian system in a stand alone programme also to do this. This can only be done by the operator.

5-3 List of FORTRAN statements

A list of the FORTRAN statements supported is given in Appendix 2. Details are given in the VARIAN FORTRAN IV manual. A list of the error messages is also given in Appendix 2, and again in the list of error messages in the Supplement from the Varian manual (p.552).
BBASIC

A version of the BASIC conversational language called BBASIC which is intended for use in the background under VORTEX is available. The present background BASIC (BBASIC) package has not proved very satisfactory and its use is not generally recommended.

If anyone wishes to use it, prior arrangements need to be made. Limited descriptive information on the package is available.
6. JOB CONTROL PROCESSOR

6-1 Purpose

The JCP is a background task that permits various software modules to be called up for background use.

6-2 Directives

Details of the JCP directives are contained in the Supplement at the end of this manual. The JCP is discussed in Section 4 (page S10). Error messages are given on p. S51. A summary of the directives is given also in Appendix 3.

Note that the directives are organised into:

(a) job initiation and termination directives
    (e.g. /JOB, /ENDJOB, /C, /MEM)

(b) I/O assignment and control directives
    (e.g. /ASSIGN, /FILE, /EEOF, /REW)

(c) language processor directives
    (e.g. /FORT).

(d) utility directives
    (e.g. /EDIT, /FMAIN, /LMGEN)

(e) programme loading directives
    (e.g. /EXEC, /LOAD, /ALTLIB, /DUMP)

These are explained in the Supplement.

6-3 Scheduling JCP

Initially JCP is scheduled by a directive on the OC (operator communication) device. There-in-after directives are to come from the SI logical unit (default assignment to teletype TY99 but can be assigned to other physical devices).

WARNING: The /FINI command transfers control back to the OC device and the system will not respond to the SI logical unit. If the OC device is not available for general use, CONTROL WILL BE LOST. DO NOT USE THIS DIRECTIVE.

If the user becomes confused as to what assignments have been made, either reassign them or type /ENDJOB or /JOB which will assign all logical units to the default list of physical devices (Table 3-4).
Notes on some JCP directives

(a) /ASSIGN. Used to interconnect logical units to physical devices (Section 3).

(b) /MEM. Used to provide extra core memory for the proposed task. Initially /JOB makes 8K of core available for background tasks. This is generally sufficient for most needs, except for /FORT compilations. The FORTRAN compiler requires more than 8K so it automatically allocates more space. However the amount of space left over is often not sufficient for the compilation of large FORTRAN programs. In which case the compilation is aborted with the error message T3. If this happens, it is necessary to use /MEM to provide more core in 4K lots. Do not allocate excessive extra core as this may not be available if there are foreground tasks.

(c) /PFILE. Used to position on the beginning of a file in a particular disc area.
7. SOURCE EDITOR

7-1 General Remarks

The source editor (SEDITION) is used for entering programs to the system and for making alterations or corrections (editing) to the program.

Details of the SEDIT commands are given in the supplement (p. S 24), the commands are also summarised in Appendix 4 and error messages are listed on p. S 54.

7-2 General Organisation

The SEDIT software is called up under JCP by the /SEDITION directive. The SEDIT software remains until another JCP command (/command) is typed.

The SEDIT software is written to read records from one source (logical unit) IN and transfer them to a second logical unit OUT (OU). While transferring from one to the other, it may also take other records from the alternative logical unit ALT(AL). This transfer is under the control of source editor commands.

The record transfer flow under SEDIT is illustrated schematically in Figure 7.1

Note that the flow is in one direction and cannot be reversed. If editing is required on some record that has been transferred, the whole procedure will have to be repeated.

![Figure 7.1 Record Transfer Flow Direction under SEDIT](image)

While the transfer is taking place, corrections may be made and lines or parts of programs inserted or removed. This provides a powerful editing facility.

7-3 Logical Units

SEDITION uses the following general logical units (Section 3)

SI - input of SEDIT commands (keyboard terminal)
SO - output of SEDIT error messages and input of error.
LO - output of SEDIT listings.

These logical units must be assigned initially by /ASSIGN under JCP before SEDIT is used.
Within SEDIT three further logical units IN, OU, ALT are involved. These are assigned under SEDIT by the AS command. The default values are IN=FI, OU=PG, ALT=BI.

7-4 Operation

Examples using SEDIT are shown in the first three examples in appendix I.

The following general procedure is used.

(i) Call up JCP and make any necessary assignments of the logical units SI, SO, LO.

(ii) Call up SEDIT and make any necessary assignments of the internal logical units OU, IN, ALT.

(iii) If editing, work sequentially using the line numbers of the input file statements (obtained from a prior listing) and the SEDIT commands.

(iv) Finally, use a JCP directive (e.g. /ENDJOB) to return control to the JCP.

7-5 Comments on Examples

SEEDIT is used in the example in Section 4-2 and example 1, 2, 3, 4, and 7 in Appendix 1.

In the example in Section 4-2 and example 1 in the Appendix, SEDIT is used to input a program either from the teletype or paper tape. This is a matter of correctly assigning the logical units, and using the FC (file copy) and LI(list) commands. In example 2 the SEDIT command is used to get a listing while copying the program from IN to OUT. In examples 4 and 7 SEDIT is used to copy a program from one disc file to another.

The editing facilities of SEDIT are demonstrated in Example 3. AL is connected to the teletype in order to insert corrections by the teletype. The SEDIT directives are input through SI also from the teletype.

Thus the SEDIT directive DE, 2 causes the first line of the program to be copied from IN to OUT, the second line to be read from IN but not transferred to OUT. The transfer stops for further SEDIT commands. The next AD, 7 causes lines 3 to 7 to be copied from IN to OUT, and then input be switched to ALT for additional program statements which are copied directly onto OUT. Input is switched back to IN where an EOF (CTRL BEL) is received. This EOF is not copied onto OUT. Line numbers throughout refer to the input file.
8-1 Introduction

The file maintenance software is a background task that manages file name directories and the space allocation of files. It can be used to create, delete or rename files and to get lists of files. It is scheduled by the JCP directive /FMAIN (Section 6).

8-2 Logical Units

SI - input file maintenance directives
LO - output file maintenance directives, directory listing, and deletion listing.
SO - output file maintenance directives, error message and error recovery directives.

8-3 FMAIN Directives

CREATE - creates new file
DELETE - deletes named file
RENAME - rename file
ENTER - add new name for existing file in directory
LIST - lists file name directory
INPUT - specifies logical limit for input of object module
ADD - adds object model from INPUT to SW.

A further directive INIT is available. This is not to be used as it will wipe all stored files.

These directives are discussed in greater detail on p. S 32 of the supplement, while error messages are listed on p. S'.55
9-1 Introduction

This is the program module that is called up to generate background and foreground tasks. It only handles 'object modules', that is programs which have been written or compiled into machine language. To be capable of scheduling a task under VORTEX it must be generated as a load module.

9-2 Directives

There are five directives only.
TIDB creates task identification block
LD load
OV overlay
LIB library search
END

These directives are discussed in greater detail on p. S 18, while error messages are listed on p. S 53.

9-3 Logical Units

SI for LMGEN directives
GO for error recovery inputs.

9-4 Explanation

The load module generator only handles an 'object module', that is a compiled program. This is usually available as a file on disc.

The JCP directive /LMGEN brings up the LMGEN software package.

The directive TIDB must be the first LMGEN directive used. It identifies the task, its scheduling and any overlay features. The LD directive specifies the logical unit from which the 'object module' is to be loaded. There may be several of these. The OV directive indicates any portion of the program that is to be overlaid. The LIB directive indicates that all the required 'object modules' have been loaded, and indicates the libraries to be searched. The END directive causes the load module to be generated, a file to be created and LMGEN to be terminated.

9-5 Example Use of LMGEN to create a foreground task.

/JOB,JOBC
JC**
/ASSIGN,LO=LP00,PI=D00K
JC**
/FILE,PI,,USER
JC**
/FORT,M

JCP
Find FORTRAN program in USER file and position on it.

FORTRAN compile, M suppresses symbol listing
1 C
2 DIMENSION IBUF(10)
3 DIMENSION ILIST(10)
4 DATA ILIST/1,3,5,7,9,2,4,6,8,10/
5 CALL V$OPIO(48,200,30000,ISTAT)
6 CALL AIRDW(10,ILIST,IBUF,ISTAT)
7 WRITE(14,1)IBUF
8 1 FORMAT(10I7)
9 STOP
10 END
Ø ERRORS COMPILATION COMPLETE
JC**
/ASSIGN,LO=CTØØ
JC**
/LMGEN
LG**
TIDB,TEST,2,Ø
LG**
LD,BO,,BO
LG**
LIB
LG**
END,FL,F
LG**
/ENDJOB
JC**

Print out of program on line printer (This is a data logging program explained in Section 10).

call up LMGEN

Task identification block. Name of task, TEST; 2, protected foreground task; Ø, no overlays.
Load; BO is logical unit where object module resides; no protection key is used; BO is file name
Library, indicates all load directives have been input

Terminates LMGEN, by creating file; FL is name of logical unit on which file containing load module resides, F protection code.
10. REAL TIME OPERATION

10-1 Introduction

The system is based on a real time computer. This means that tasks can be interrupted.

A VORTEX software package, the real time executive (RTE) controls the running of tasks, their interruption and resumption in response to their priority levels and other interrupts. This is termed priority interrupt control.

For example, if a new task is entered with a higher priority than the present task, the RTE will suspend the present task, allow the new task to proceed (subject to interruptions with higher priority) until it has no further immediate need of the system facilities (e.g. it has finished or it is waiting for transfer of data), and then the RTE will resume the prior task (provided it still has the next highest priority).

The RTE services all interrupts which occur. These may be of several types:

(i) System internal interrupts (Section 10-2).
(ii) External (hardware) interrupts (Section 10-3).
(iii) Operation requests on OC device (Section 10-4).
(iv) FORTRAN statements using RTE facilities (Section 10-5).

10-2 System Internal Interrupts

Within the organisation of the system, tasks will be interrupted for:

(i) memory protection, e.g. when a program tries to write over a protected part of the memory.

(ii) power failure. When a power failure is detected (line voltage starts to drop) the current task is interrupted and the contents of the computer are saved, so that the program can be continued after reapplying power.

(iii) real time clock. The real time clock is the basis for timekeeping in the system. It interrupts operations to update the time of day and to check for any tasks scheduled.

These interrupts have the highest priority in the machine, with priority in decreasing order (i), (ii), (iii). These interrupts are automatic and are not of direct concern in programming.

10-3 External (Hardware) Interrupts

External interrupts come from hardware items (e.g. the teletype controller indicates it has finished printing a character, and hence further transfer of data is possible). The hardware within the computer for handling these interrupts is a PIM (priority interrupt module). This is a circuit board within the machine. Each PIM handles eight interrupt lines and has an 8 bit register. When an interrupt is requested on one of the lines, this results in a change in that bit in the register. These registers are scanned and so the interrupt is effected.
For present purposes, we are not interested in interrupts from hardware items. If we wish to throw a switch to initiate a data logging experiment, this will be handled by a software command. Our immediate interest therefore is in the software directives.

10-4 Operator Communications (OC) Requests

Special requests typed in on the operator communications (OC) device will affect real time operations.

These requests are ONLY effective on this OC device. This device is teletype TY10. It is not available for use after hours.

There are a number of OC requests. Some requests have a considerable potential for causing inconvenience and we would request that you DO NOT USE ANY OC REQUESTS OTHER THAN THE FOLLOWING:

; SCHED - schedule foreground task
; TSCHED - time schedule foreground task
; ATTACH - attach foreground task to PIM line
; RESUME - resume task
; TIME - enter or display time of day
; DATE - enter date
; ABORT - abort task.

All such commands begin with a semicolon.

Details of these requests are given on pS37, while error messages are listed on pS56.

An OC request is the means of scheduling the running of tasks in the foreground (Section 10-6).

10-5 FORTRAN Statements Using RTE Facilities

Special FORTRAN statements (e.g. CALL SUSPND) can be included in FORTRAN programs. These statements reference RTE macros which affect real time operation of the task. These statements can be used to suspend execution, overlay a section of the program, schedule another task, etc.

The available statements are:

CALL SCHED - schedule a task
CALL SUSPND - suspend a task
CALL RESUME - resume a task
CALL DELAY - delay a task
CALL LDELAY - delay and reload
CALL PMSK - store PIM mask register
CALL TIME - time of day
CALL OVLAY - overlay segment
CALL ALOC - allocate reentrant stack
CALL EXIT - exit from task
CALL ABORT - abort a task

These calls are summarised on pS1 and then explained in detail, with the error messages listed on pS46. The detailed explanations cover the RTE macro first then the FORTRAN call.
Control of Real Time Operation

You have been using a real time system for the background computations already discussed in the earlier chapters. Because it has been a background task, and thus of the lowest priority (levels 0 and 1), serious concern regarding real time operation was unnecessary.

It is possible to include the FORTRAN real time calls in programs running in the background (10-5) but again these should cause no concern because of their low priority. Some of the calls listed are not applicable for background tasks (refer pS1).

However, when programs are being run in the foreground, a knowledge of real time operation is essential. Only programs in object form (e.g. compiled FORTRAN programs) can be run in the foreground. The object program is transferred to the foreground using the LOAD MODULE GENERATOR (Section 9).

Thus to run tasks in the foreground they have first to be converted from 'object modules' by LGGEN to 'load modules'. These are resident in an appropriate library. A task is initiated for present or future execution (and with its priority level) by the requests typed in on the operator communications (OC) device (Section 10-4).

For example, consider the load module generated in the example in Section 9-5, now resident in the foreground library FL. For "immediate" (depending on other priorities) running of this program in the foreground with priority level 5, type in on the OC device (TTY10),

;SCHED, TEST, 5, FL, F

To schedule it to run at 1900 hours that evening (other priorities willing) with priority level 8 use the time schedule directive thus, -

;TSCHED, TEST, 8, FL, F, 1900

Programs which would conveniently run in the foreground are those running for some considerable time, but only requiring a small portion of the computer facilities for that time (e.g. data logging or control operation).

Priority Levels

The system is organised about 32 priority levels (0 through 31). The lowest levels 0 and 1 are fixed for the background. Certain predetermined tasks have already been given high priority levels in the foreground and high priority levels (>20) should be avoided.

Initially users should consider priority levels from 2 to 20. These levels have not been used otherwise by the system. Several tasks may have the same priority level.
11-1 INTRODUCTION

The storage of programs as files is quite straightforward, by the use of SEDIT (Section 7) and general organisation under FMAIN (Section 8). With these, program files may be copied, edited, created, renamed, deleted, etc.

However a file may also be used to store data generated in the running of a program, or to provide data required during the running. For example, results from data logging may be stored in a file, and in a subsequent program be recovered for further calculation. In other applications a file may be used as a data bank to be continually updated or interrogated by various programs. This is a dynamic or run-time use of input and output to files. The FORTRAN directives listed below can be used to enter and recover data from files for such applications.

11-2 DIRECTIVES

The following are FORTRAN statements which may be used within FORTRAN programs to access files during program running.

CALL W$OPEN - open a file for standard record transfer
CALL W$CLOSE - close a file after standard record transfer
CALL W$OPENB - open a file for blocked record transfer
CALL W$CLOSEB - close a file after blocked record transfer
CALL IOCHK - input-output check

These statements are discussed in greater detail in 11-5. Error messages are those listed with the FORTRAN runtime errors (p552) and I/O errors (p548).

11-3 EXPLANATION OF OPERATION

We are concerned here with files on disc. These are located on portions (partitions) of the disc designated by various logical unit numbers (lun) e.g. 15, 16 (refer table 3-4).

A file is composed of records. There is no limit on the number of records a file may contain, provided sufficient disc space has been made available.

A record is a string of a number of words. With the 16 bit word of the Varian, one word can contain 2 characters (as 8 bit ASCII). Numbers are not stored separately but also as characters, i.e. digit by digit as 8 bit ASCII.

The transfer of data between the working core and the file on the disc partition takes place through a buffer (Figure 11.1).

![Figure 11.1 Record transfer core-buffer-disc](image-url)
The transfer between the disc and the buffer takes place with records of 120 word size. This is termed the standard record or physical record. It is the basic unit in all file storage and file manipulation on disc.

However the transfer of data between the program (running in core) and the buffer (Figure 11.1) may take place with records of any arbitrarily selected size. These records are transferred to the buffer until it contains at least 120 words (or an EOF terminating transfer). A 120 word standard record is then transferred to the disc file. The remaining buffer words are shifted to the top of the buffer and further transfer from the program proceeds until it again equals or exceeds 120 words. Transferring from file to the program works in reverse. Records of the designated length are transferred from the buffer to the program until no further complete record remains. The residual words are shifted to the top of the buffer and the next 120 word standard record is transferred from disc. Buffer to program transfer recommences.

There are two distinct possibilities in selecting the record size for program-buffer transfer, -

(a) standard records
(b) blocked records (record size, recas ≠ 120).

(a) The standard record is of 120 words (240 character) size. With this, the record sizes for transfer between program and buffer, and buffer and disc, are identical, and transfer operations are simple. The commands V%OPEN and V%CLS are used to open and close files when a standard record size is required. Once a file is open, data may be entered or recovered by suitable FORTRAN WRITE and READ statements.

The commands for standard record transfer are simpler than those for blocked transfer, since no decisions are required about record size and buffer size. However as the standard record size is often very much larger than the data record it contains, it is not an efficient use of disc storage space.

(b) The blocked record may have a record size (recas) of any arbitrarily selected number of words. The calls V%OPNB, V%CLS are used to open and close files when blocked record transfer between program and buffer is required. Again data is entered or recovered from the opened file by suitable FORTRAN WRITE and READ statements.

With the blocked record command V%OPNB, extra information is required on, -

(i) the record size (recas) chosen,
(ii) space allocation for the buffer. This is to provide sufficient room to handle the accumulation due to the different record sizes.
(iii) a read-before-write flag option.

These are discussed in greater detail in Section 11-5.

11-4 File Control Block (FCB) Array

When a file is opened using either V%OPEN (for standard record size) or V%OPNB (for blocked records) the computer constructs a file control block (FCB) for its own internal monitoring. This contains the file name, its protection code and other information. The file opening commands V%OPEN, V%OPNB, therefore reference a small integer array, the FCB array, which has
to be filled with this information by the user. This array is 13 words long for V$OPEN, 14 words for V$OPENB, and must be DIMENSIONed initially. Locations (words) 8, 3, 10 of this array must contain the name of the file to be opened. This can be inserted by a DATA statement (2 characters to a word) – see examples in Section 11-5. Location 3 must contain the protection code of the file. Location 4 contains the current pointer location, giving the present record position in the file. Each READ or WRITE operation automatically updates this pointer position by 1 (i.e. moves to next record position). The pointer position may be altered during transfer operations by changing the number in location 4 of this file control block array – see examples in Section 11-5.

11-5 Details of Commands

(a) For standard records transfer, V$OPEN, V$CLOSE

CALL V$OPEN(fun,lun,name,mode)
  e.g. CALL V$OPEN(22,16,IFCB,0)

CALL V$CLOSE(fun,mode)
  e.g. CALL V$CLOSE(22,1)

where fun is the name or number of the FORTRAN unit, defined in a DATA statement or Hollerith character string.

lun is the name or number of the file logical unit, defined in a DATA statement or Hollerith character string.

name is the name of the 13-word array containing the file name and the protection code.

mode is the mode of the I/O-control macros. Mode = 0 (default value) or 1. When opening a file 0 = rewind, 1 = no rewind; when closing a file 0 = no update, 1 = updated. Rewind sets the pointer position back to the beginning of a file. Updating ends the file after the present location.

Call V$OPEN creates an association between the file named in name and a FORTRAN logical unit, fun. It creates a FCB for internal monitoring using the information in the file control block array, name. CALL V$OPEN overrides any prior /FILE assignments. CALL V$CLOSE dissolves the association formed by V$OPEN.

Example

To read records 1 and 61 from a file FIL on logical unit 16 having no protection code, and then close the file.

C

PROGRAM TO READ FILE
DIMENSION IFC(13)
DATA IFC(8),IFC(9),IFC(10)/2HFI,2HL,2H /
CALL V$OPEN(22,16,IFC,0)
READ(22,80)I1,I2,I3,I4
IFC(4)=61
READ(22,80)J1,J2,J3,J4
CALL V$CLOSE(22,0)
80 FORMAT(4I4)
A 13-word integer FCB array (in this case IFC) must first be dimensioned. Locations 8, 9, 10 are filed with FIL, the name of the file to be opened using a DATA statement. If a protection code were required this could have been loaded in IFC(3) by a similar command.

The file is opened by V$OPEN referencing a FORTRAN logical unit number of 22, which must be used in future WRITE and READ statements for this file, a lun of 16 (for the disc area), the array name IFC of the file control block array, and mode = 0 (on opening to rewind to first record on file). The first record is READ on calling FORTRAN logical unit 22 under FORMAT 80, and storing four values (actually 8 words) from this record in I1, I2, I3, I4. After this READ the current position pointer moves to record 2. Location 4 in the IFC array is then changed to 61 which changes the current position pointer in FIL to 61 for the next READ statement. This copies four integer values into J1, J2, J3, J4. The file is closed with no updating, since the further records may still be wanted in future use of the file.

(b) For blocked records transfer, V$OPEN, V$CLSBL

* CALL V$OPEN(fun, lun, name, mode, reoss, buff, rbwfl)
  e.g. CALL V$OPEN(22, 15, IFCB, 0, 10, JBUFF, 1)

* CALL V$CLSBL(fun, mode)
  e.g. CALL V$CLSBL(22, 0)

where

fun is the name or number of the FORTRAN unit, which is defined in a DATA statement or Hollerith character string.

lun is the name or number of the logical unit, which is defined in a DATA statement or Hollerith character string.

name is the name of a 14-word FCB array.

mode is the mode of the I/O control macros. Mode = 0 (default) or 1. When opening a file 0 = rewind, 1 = no rewind; when closing a file 0 = no update, 1 = updated. Rewind sets the pointer position back to the beginning of a file. Updating sets a new end of file mark after the present location.

reoss is the logical record size measured in number of words. The record size should be chosen commensurate with your needs. For example if filing I4 data one to a record, reoss = 2, for input/output using the 72 character teletype, reoss = 36, for use of 132 character line printer, reoss = 66.

buff is the name of a blocking buffer array. This array must be large enough to handle the accumulation due to the different record sizes of program-buffer and buffer-file transfers. The following relations allow the buffer size to be estimated:

\[ \text{reoss} < 120 \]

\[
\begin{align*}
\text{R(120/reoss)} & \quad \text{Size of Array buff} \\
= 0 & \quad 120 \text{ words} \\
\ne 0 & \quad 240 \text{ words}
\end{align*}
\]
recsz \geq 120
\begin{align*}
R(\text{recsz}/120) & \quad \text{Size of Array buff} \\
= 0 & \quad \text{recsz} \\
= 1 & \quad 120 \times (1 + Q(\text{recsz}/120)) \\
> 1 & \quad 120 \times (2 + Q(\text{recsz}/120))
\end{align*}

R = \text{remainder}; \quad Q = \text{quotient}.

This integer blocking buffer array must be DIMENSIONed initially.

rbwfl is the read-before-write flag. On a WRITE operation where recsz is not a multiple of 120 words, data on the file can be overwritten unless a read-before-write is performed. In some situations, such as initial file creation in a strictly sequential fashion, this is unnecessary and slow. The parameter rbwfl allows the user to select this feature. If rbwfl = 0, read-before-write is disabled. Any non-zero value enables read-before-write.

CALL VSOPNB creates an association between the file named in name and a FORTRAN logical unit, fnum. It creates a FCB for internal monitoring using the information on the file control block array, name. Note that this array is now of size 14. The command also creates an extended buffer array for blocked records buffering. CALL VSOPNB overrides any prior /PFILE assignments. CALL VSCLSB dissolves the associations formed by VSOPNB.

Example To read data from the teletype and write into a file BLFILE on logical unit 15 with no protection code. Record lengths of 10 words will be used.

```c
C PROGRAM TO WRITE BLOCKED RECORDS INTO FILE
DIMENSION IFCB(14),IBUFF(120)
DATA IFCB(8),IFCB(9),IFCB(10)/2HBL,2HFI,2HLE/
CALL VSOPNB(4,15,IFCBB,0,10,IBUFF,1)
DO 100 I=1,20
READ(13,80)VALUE
80 FORMAT(F20.5)
100 WRITE(4,30)VALUE
90 FORMAT(F20.5)
CALL VSCLSB(4,0)
```

The integer arrays for FCB and buffer purposes must be dimensioned initially. The size of IBUFF is given by the formulae quoted. The file name BLFILE is inserted in locations 8, 9 and 10 of IFCBB. The blocked record file opening command VSOPNB associates BLFILE on logical unit 15 with the FORTRAN logical unit 4. The file is rewound on opening so entry starts at the first record position. A value is read from the teletype and written as the first 10 word blocked record in IBUFF. This continues until 12 blocked records have been stored in IBUFF. A 120 word physical record (of 12 blocked records) is then transferred to BLFILE from IBUFF, and the operation continues. The file is closed without updating.
(c) **Input-output check IOCHK**

This allows the user to check whether the last transfer of data was completed normally.

The FORTRAN input-output (I/O) program allows a program to detect I/O errors and end-of-file or end-of-device conditions. Status of a READ or WRITE operation is available immediately after the operation is complete and before another I/O operation is executed. This status can be checked by executing a subroutine or function call in the form:

```
CALL IOCHK(status)
```

where `status` is the name of an integer variable which is to receive the result of the status check.

If the last I/O operation had been completed normally, the value of zero will be returned. If an error had occurred, the value minus one is returned. If either an end-of-file or an end-of-device had occurred, the value positive one will be returned.

The status may be checked and the result tested in a single statement by use of the form:

```
IF (IOCHK(status)) label(1), label(2), label(3)
```

where `status` is the name of an integer variable which receives the result of the status check. A value of zero indicates normal completion. A positive non-zero value indicates an error. A negative non-zero value indicates EOF or EOD.

`label(1)` is a statement label to which control is transferred, if an I/O error occurred.

`label(2)` is a statement label to which control is to be transferred if the operation was completed normally.

`label(3)` is a statement label to which control is transferred, if an end-of-file or end-of-device was encountered.

If the program does not check the status of a READ or WRITE operation, FORTRAN will abort execution of the task upon the next entry to the runtime I/O routine. At that time the diagnostic message will be output to the System Output device. Any data which is input to a READ in which an error occurred will be invalid. After a call to IOCHK is executed, any error status is reset and the program may proceed with additional input and/or output.

For example, refer use in graphics example 6 (p13-20).

### 11-6 Comments on Commands

(a) While a file is open, it may be rewound or backspaced by manipulating location 4 in the FCB array, e.g.

```
IFCB(4) = 1    rewind
IFCB(4) = IFCB(4) - 1    backspace
```
(b) It is possible to operate directly on a file (in the background ONLY) without these special file opening commands. This is done by the /PFILE directive. This method is given for completeness, but its general use is not recommended.

For example,

```
/ASSIGN,PI=16
/PFILE,4,,MYFILE
/FORT
$ *
READ(4,\cdot\cdot\cdot)
```

The /PFILE locates the PI unit at the beginning of the file FILE on logical unit 16 (with no protection code). In the later FORTRAN program a READ statement calling the logical unit number of PI (4) as its FORTRAN unit number will access records in this file starting from the beginning and progressing sequentially.
12 DATA LOGGING AND CONTROL

12-1 Hardware

The signal processing hardware available on the system has been discussed in the description of the system (part I of manual). The system has

- 16 analog inputs (± 1 volt)
- one 16 bit digital input (not available under VORTEX)
- 4 analog outputs (± 10 volts)
- two 16 bit digital outputs
- 16 status inputs
- 16 control outputs

The majority of these connections are available on the IFC (interface console) front panel. The rest (one set of 16 bit digital output, 8 status inputs and 8 control outputs) must be connected at the rear of the machine.

12-2 FORTRAN statements for Input/Output Calls

FORTRAN subprograms calls are available which simplify process input and output applications.

These are,

CALL V$OPIO - opens input-output
CALL AISQ - analog input sequential
CALL AIRD - analog input random
CALL AO - analog output
CALL DI - digital input (not relevant)
CALL DOM - digital output momentary
CALL DOL - digital output latching

Each of the last six calls can use a 'wait' option. This is invoked simply by adding a W at the end of the call, eg. CALL AISQW, CALL AIRDW, CALL AOW, etc.

These calls are discussed in detail on p S42, with errors on p S44.

As well, there are two further calls-

CALL STAT - status input
CALL PULSE - control outputs

These are discussed in Section 12-3.

12-3 Control and Status Line Operation (PULSE, STAT)

The control outputs and status inputs are accessed by special software instructions of our own, hence there is no description of them in the VORTEX manual.

These devices are concerned with a single bit of information (0 or 1).
(a) **CONTROL OUTPUT**

The sixteen control outputs are operated by flip-flops which are set by the FORTRAN call

```
CALL PULSE (N)
```

where \( N \) = channel number of the control output.

There are three connections for each control output (labelled \( OUTPUT \), \( GROUND \) and \( RESET \)). A control output may be likened to a switch. The above call connects (shorts) \( OUTPUT \) to \( GROUND \). Connecting \( RESET \) to \( GROUND \) will disconnect (isolate) \( OUTPUT \) from \( GROUND \). The maximum current rating is 300 mA. The outputs operate on a 5 volt system.

Figure 12-1 shows how two control outputs might be used to switch a small lamp on and off.

![Diagram of control outputs](image)

Figure 12-1  Use of control outputs to switch lamp on and off.

Initially, consider \( l \) switched 'off' (i.e. \( OUTPUT \) is not connected to ground). No current will flow so the lamp will be off. Execution of CALL PULSE (1) will switch \( l \) 'on' (i.e. connect \( OUTPUT \) to ground) and the lamp will light. CALL PULSE (2) will ground \( OUTPUT \) 2 and thus \( RESET \) \( l \), which will disconnect \( OUTPUT \) and \( GROUND \) on \( l \), so the lamp goes out. The interconnection between \( GROUND \) and \( RESET \) on 2 results in an output pulse of 0.2 \( \mu \)s duration. This is sufficient to reset \( OUTPUT \) \( l \) but leaves 2 still off. Thus CALL PULSE (1) will switch lamp on again, and CALL PULSE (2), off.

(b) **STATUS INPUT**

A signal line may be connected to a status input so that the computer can sample it to see if the signal line is high (i.e. has a voltage \( 2\frac{3}{4} \) to 5V) or is grounded (voltage 0 to \( \frac{1}{2} \)V)

The FORTRAN call

```
CALL STAT (N, IVAR)
```

where \( N \) = channel number of status input

- \( IVAR \) = integer variable, set = 0 if line is high (false)
- or 1 if line is low (true)

For example in Figure 12-1, a line could connect \( OUTPUT \) \( l \) to a status \( INPUT \) channel (5, say). The command CALL STAT (5, JJ) could be used in the program to sense if the lamp were on or off. If \( JJ=1 \) the line is low (grounded) and the lamp is on, if \( JJ=0 \) the lamp is off.
Organisation of VORTEX Process I/O Calls

(i) V$OPIO

To use the process I/O calls listed in section 12-2 (except for CALL PULSE and CALL STAT) the CALL V$OPIO must first be made.

CALL V$OPIO associates the device address and logical unit number with the sample time required. Table 12-1 gives the device address, logical unit numbers and channel numbers for the various devices.

<table>
<thead>
<tr>
<th>Process I/O</th>
<th>Device address, lun</th>
<th>Channel numbers (decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALOG INPUT</td>
<td>48,200</td>
<td>1 to 16</td>
</tr>
<tr>
<td>ANALOG OUTPUT</td>
<td>49,201</td>
<td>0 to 3</td>
</tr>
<tr>
<td>DIGITAL OUTPUT</td>
<td>41,202</td>
<td>8, 9 (16 bit)</td>
</tr>
<tr>
<td>DIGITAL INPUT</td>
<td>not available</td>
<td></td>
</tr>
</tbody>
</table>

Table 12-1 Process I/O device information

The sample time is the desired duration between successive data points being inputted (input only). For output this variable is ignored. The sample time in µs should be greater than 20 (transfer time to array) and should not exceed 32767 (the maximum integer number allowed). If longer sample times are required they may be generated by a DO loop incorporating a CALL DELAY (Section 10-5). This CALL is only available in foreground operation.

(ii) W option

As indicated in section 12-2 a 'wait' option is available. It is invoked by adding W to the end of the CALL name.

The W stops further execution of your FORTRAN program until the process I/O CALL command is complete. Without the W it is possible that the system will continue with subsequent parts of the program while the I/O operations are being carried out. The W option on these calls is definitely recommended.

(iii) Other variables

The variables COUNT, PLIST, IBUF, OBUF, MARK and SIAT are not explained in detail in the text.

COUNT Number of data values to be transferred. This need not be equal to the number of channels involved.

PLIST Name of integer array containing information on channels to be accessed.
IBUF, OBUF  Names of integer buffer arrays in which data values are to be entered or outputted. These arrays should be DIMENSIONed large enough to contain COUNT values.

MASK  Name of integer array, similar to IBUF and OBUF, to contain bits to be changed by DOL

STAT  Name of integer variable in which status of call is to be stored (refer p S42)

For all calls except AISQ, AISQW (sequential sampling) PTLIST contains the numbers of the channel (Table 12-1) to be accessed in order. The same channel number may be repeated throughout the list, if more frequent accessing of that channel is desired. For AISQ and AISQW PTLIST contains only the number of the last channel to be inputted. Here inputting starts from the first channel on that device and progresses sequentially to the last channel indicated.

If COUNT is greater than the number of channels indicated, it will progress through the sequence again. COUNT is the number of data values involved, not the number of cycles for each channel.

12-5 Calibration of Analog Input and Output

The analog inputs are calibrated for a ±1v input which on conversion gives 13 bit digital numbers from -4096 (-1 volt) to + 4095 (+0.99975v).

The analog outputs are driven by 12 bit digital to analog converters to give ±10v outputs. Thus only digital numbers in the range -2048 (-10v) to +2047(+9.995v) will be accepted correctly. Numbers outside this range will cause complications.

The two 10 turn helipots are calibrated to provide for 0 to ±10v for checking purposes.

12-6 Examples of FORTRAN programs for data logging and control

(a) To sample analog input channels: 1, 3, 5, 7, 9, 2, 4, 6, 8, 10 once, in that order, one every 30 ms. This is a non-sequential (random) order so AIRDW is required. (This is the example used in section 9-5)

```
DIMENSION IBUF (10)
DIMENSION ILIST (10)
DATA ILIST/1,3,5,7,9,2,4,6,8,10/
CALL V$OPIO (48,20,300000,ISTAT)
CALL AIRDW (10,ILIST,IBUF,ISTAT)
WRITE(14,1)IBUF
1 FORMAT(10I7)
STOP
END
```

The integer file ILIST is loaded with the channels numbers (Table 12-1) to be scanned (in order), by a DATA statement. CALL V$OPIO associates the analog input section (device address 48, logical unit number 200) with the sample time (30000 microseconds). ISTAT will contain the status of the input transfer (p S42). CALL AIRDW will cause 10 data
values to be taken according to the sequence of channels in ILIST and the values to be stored in IBUF. The W (wait) option is used so that the sampling will be completed before the program proceeds further.

(b) To output the first three words of OBUF to analog output channels 3, 0, 1 in that order (a control situation).

```
INTEGER STAT, PTLIST, OBUF
DIMENSION OBUF(3), PTLIST (3)
DATA PTLIST/3,0,1/
CALL V@POIO (4,0, 2,1, 0, STAT)
CALL AOW (3, PTLIST, OBUF, STAT)
```

This example is similar to Example 2 on p 842, with device information (Table 12-1) selected for our system. The time in V@POIO is ignored for output so 0 has been used.

(c) To output one word (16 bits) of digital output from OBFF to channel 8. Each of the 16 digital output channels behaves in a similar way to the control outputs (section 12-3a) and can be used to switch on or off appropriate devices.

```
INTEGER OBFF
DIMENSION ILIST (1), OBFF(1)
DATA ILIST/8/
CALL V@POIO (4,1,2,0, ISTAT)
CALL DOM (1, ILIST, OBFF, 0, ISTAT)
```
13-1 Introduction

The graphics software package consists of a set of FORTRAN statements. These allow graphs to be drawn and data points to be plotted on the CRT screen.

The graphics software is based on the TEKLOT 'PLOT 10' graphics package for the Tektronix Model 4010 CRT with modifications to suit the Varian VORTEX system.

The commands to operate the graphics are written as FORTRAN statements within a normal FORTRAN program. This is to be compiled and executed in the normal way.

13-2 List of Commands

General

CALL START - initialises the graphics routine
CALL ERASE - erases the screen
CALL HOME - positions cursor to upper left hand corner
CALL TPAUSE- allows pause

Mode Selection

CALL ALPHA - switches to alphanumeric mode
CALL VECTOR- switches to vector mode
CALL POINT - switches to point plot mode

Axis Manipulation Routines

CALL SCALE - fixes origin and scaling factors
CALL ROTATE- rotates axes
CALL WINDOW- trims to window size
CALL AXIS - draws axes

Plotting Routines

CALL TPILOT - plots points
CALL DELTA - incremental plotting

Cursor Input Routines

CALL CURSIS - inputs cursor coordinates
CALL SETGRD - plots grid
CALL GRID - uses cursor with grid

Software Character Generator

CALL WRITEX - draws characters with current scaling
CALL WRITEX - draws characters with local scaling
CALL XWRITE - converts numbers to correct form (ASCII) for plotting.
13.2 Scope of Graphics Software

The graphics routines allow the user to think of the CRT screen as a 'window of vision' on portion of the plane of the plotted results. Using commands, this window may be translated to any part of the plane and rotated, with the further ability to change the magnification within the viewed area.

This may be accomplished either by referring to coordinates on the plot plane, or by visually inspecting the results and using the position of a cursor cross lines to input locations.

The software allows:

- plotting to be done either absolutely (relative to an origin) or incrementally (relative to the previous positions of the beam)
- axes and grids (linear or log) to be drawn
- characters to be drawn to any size and orientation
- information to be input from the position of the cursor

13.4 Hardware Organisation

The screen of the 4010 is divided into points, 1024 horizontally (counted 0 to 1023) and 781 vertically (counted 0 to 780) with the origin (0,0) at the bottom left hand corner as shown in Figure 13.1,

![Figure 13-1 Organisation of Visual Display Screen](image)

Plotting is done by drawing a straight line vector from the current position of the CRT beam to the new designated point. This new point then becomes the current position. To generate curves a series of short straight vectors have to be used. The smallest change possible is one raster unit, as the beam only moves from one raster point to another. This provides the ultimate resolution and smoothness of the plots. In drawing a vector, the designated point can be referenced absolutely (with respect to a prescribed origin) or incrementally (with respect to the current beam position).

The 'window of vision' can cover all or portion of the screen area. Any plotting extending beyond the window will be clipped at the window edge.

Using certain commands, (CURSIS,GRID) the cursor cross wires will be seen. These cross wires may be positioned using
the two thumbwheels on the keyboard.

13-5 Software Organisation

The CRT screen may be used in two broad modes of operation, either alphanumeric mode or graphics modes.

(1) alphanumeric mode (ALPHA). Here the 4010 behaves as a teletype, and alphanumeric information coming from the computer is displayed on the screen from the top of the page down.

This is the normal mode of operation. The terminal is automatically placed in this mode when it is switched on. Pressing a carriage RETURN or the PAGE key will reset the terminal to this mode. When in alphanumeric mode, the current character position is signalled by the location of the pulsating cursor. In ALPHA mode, the keyboard has its usual function. Always leave the terminal in ALPHA mode.

(2) Graphics Mode. In graphics mode the screen may be used to plot data, outputting either vectors (lines) or points, or inputting the position of the cursor. This gives three different modes of graphics operation,-

(a) VECTOR, vector plot, the first graphics mode. This mode is required when vectors (straight lines) are drawn.

(b) POINT, point plot, the second graphics mode. This mode is used to cause a point to appear at the new specified position.

(c) cursor input, the third graphics mode. This mode is called automatically as required in commands associated with the inputting of locations from the cursor crosswires.

The ALPHA mode and the first two graphics modes (VECTOR, POINT) must be called by commands, but as the fourth is handled automatically, no call is necessary.

In graphics mode the keyboard may have special functions.

It is important to switch to the correct mode for the application, otherwise the commands will not perform correctly.

Note the differences in the initial position of the beam in ALPHA and GRAPHICS modes. In the graphics modes, the origin is at (0,0) as shown in Figure 13-1. However the "home" position in ALPHA (alphanumeric) mode is situated at the upper left hand corner of the page (0,767). In other words, in ALPHA mode the screen is a 'page' and output starts at the expected point, while in the graphics modes the usual right hand positive x and y axes are considered, and the origin is at (0,0). In the software, these two conventions are maintained separately. As a result the current position in the graphics mode is not affected by intervening printing that occurs in alphanumeric mode.
13-6 Details of Commands

These commands are FORTRAN statements and must be used in FORTRAN programs.

13-6.1 General Commands

(a) CALL START

PURPOSE: Initializes the TEKPLLOT routines.

USAGE: CALL START

DESCRIPTION:

START should always be the first subroutine to be called in any program; it should be called only once. START is used to initialize the TEKPLLOT routines.

After calling START, the 4010 will be initialized to:

1. Alphanumeric mode
2. Current graphics position (0,0)
3. 1 raster unit/scaled unit on both axes
4. An origin of (0,0)
5. Rotation of 0 degrees
6. Normal character font
7. A window size equal to the screen size

START also determines delay intervals required for the ERASE routine.

(b) CALL ERASE

PURPOSE: To erase the screen.

USAGE: CALL ERASE

DESCRIPTION:

The ERASE subroutine erases the screen, places the terminal in alphanumeric mode, and positions the alpha cursor to the home position. ERASE does not affect the current graphics position.

Since a time delay of 750 milliseconds is required while the erase cycle goes to completion, ERASE also generates an appropriate delay. This is accomplished by sending the 4010 a sequence of IDLE characters following the control sequence that causes erasure.

(c) CALL HOME

PURPOSES: Position the alpha cursor to the upper left corner of the screen

USAGE: CALL HOME
DESCRIPTION:

HOME places the terminal in alphanumerics mode at the home position. The current graphics position is not affected. Home position is defined to be the raster point (0,767).

(d) CALL TPAUSE

PURPOSE: To create a pause during program execution.

USAGE: CALL TPAUSE

DESCRIPTION:

The TPAUSE routine provides the user with a convenient way to cause an indefinite pause during program execution. TPAUSE places the terminal in alphanumerics mode at home position, then reads one character from the terminal, discards that character, and exits. The routine is intended for a user who wishes to pause at home point in his program in order to inspect his plotted output, and then resume execution by typing any character. TPAUSE does not affect the current graphics position.

13.6.2 Mode Selection Commands

These commands are used to change the mode of operation of the CRT (refer 13.5)

(a) CALL ALPHA

PURPOSE: To place the 4010 into alphanumerics mode.

USAGE: CALL ALPHA

DESCRIPTION:

Calling ALPHA places the terminal in alphanumerics mode.

If the terminal was already in alphanumerics mode, then calling ALPHA will have no apparent effect. If the terminal was in one of the graphic output modes (VECTOR or POINT), then the alpha cursor will appear at the current graphics position.

(b) CALL VECTOR

PURPOSE: To place the 4010 in vector mode.

USAGE: CALL VECTOR

DESCRIPTION:

Calling VECTOR places the 4010 in vector mode and positions to the current graphics position. All subsequent plotting in this mode will be in the form of smooth lines connecting each specified point.
(c) CALL POINT
PURPOSE: To place the 4010 into point plot mode.
USAGE: CALL POINT

DESCRIPTION:
Calling POINT places the terminal in point plot mode and positions to the current graphics position. All subsequent plotting in this mode will be in the form of individual points that are drawn at each specified coordinate.

13.6.3 Axis Manipulation Commands
These commands are used to select origin, scale, orientation, window size and to draw axes.

(a) CALL SCALE
PURPOSE: A call to SCALE determines scale factors and established an origin for any subsequent plotting. If SCALE is not called, then the indicated default values are assumed.
USAGE: CALL SCALE (XFACT, YFACT, XORG, YORG)

PARAMETERS:
XFACT - the number of raster units per scaled unit on the x-axis. The sign of XFACT determines the direction of the x-axis, i.e., if XFACT is negative, the positive direction of the x-axis is to the left. Default value = 1.

YFACT - the number of raster units per scaled unit on the y-axis. The sign of YFACT determines the direction of the y-axis. Default value = 1.

XORG - the origin on the x-axis expressed in raster units. Default value = 0.

YORG - the origin on the y-axis expressed in raster units. Default value = 0.

DESCRIPTION:
When SCALE is called, XFACT and YFACT act as multiplicative factors that determine the conversion from scaled units to raster units. Raster units refer to the point coordinate units of the screen (Fig 13.1) while scaled units refer to the units in which data is to be calculated for plotting. XORG and YORG act as linear displacements from raster unit (0,0). For example, the physical origin of 4010 is raster unit (0,0) at the lower left corner of the screen. To establish a new origin at the center of the screen, the user may

CALL SCALE (1.,1.,512.,390.)

This new origin then remains for all subsequent plotting or until the next call to SCALE.
(b) CALL ROTATE

PURPOSE: By calling ROTATE, the user declares himself to be working in a new transformed axis system that is rotated the specified number of degrees with respect to the Cartesian system of the screen. The center of rotation is the user's origin, scaled unit (0,0).

USAGE: CALL ROTATE (DEG)

PARAMETERS:

DEG - the number of degrees (not radians) of rotation from the orthogonal system of the screen. Default value = 0. Positive rotation is in a counter clockwise direction.

DESCRIPTION

After calling ROTATE, all further plotting will occur in the new transformed coordinate system until the next call to ROTATE. For example, if the user executes the following

\[
\begin{align*}
\text{CALL SCALE (1.,1.,512.,380)} \\
\text{CALL ROTATE (90.)} \\
\text{CALL TPLT (0.,0.,0,0)} \\
\text{CALL TPLT (0.,100.,1,0)}
\end{align*}
\]

then the first call to TPLT will establish a current graphics position at the user's logical origin, raster point (512,380). The next call to TPLT causes a line to be drawn parallel to the user's y-axis. Since the user's axis system has been rotated 90 degrees, this line will appear to be parallel to the screen's x-axis.

(c) CALL WINDOW

PURPOSE: The WINDOW subroutine is used to set the window size for all subsequent plotting. Any plotting that extends beyond the current window will be clipped at the window margin.

USAGE: CALL WINDOW (XMIN, YMIN, XMAX, YMAX)

PARAMETERS:

XMIN - the minimum x value of a rectangular window. XMIN should be given in scaled, rotated units.
YMIN - the minimum y value of the window. YMIN should be given in scaled, rotated units.
XMAX - the maximum x value of the window, specified in scaled, rotated units.
YMAX - the maximum y value of the window specified in scaled, rotated units.
DESCRIPTION:

WINDOW converts the user's parameters to raster units and saves them in the terminal status area for future use. If the user's XMAX parameter turns out to be smaller than XMIN, then these are interchanged. Similarly, if YMAX is less than YMIN, then these are interchanged.

If the WINDOW parameters specify a window that is larger than the 4010 screen size, then the designated window is changed to the screen size.

Because window parameters are specified in terms of scaled units, parameters may be set by visual inspection of a plot using the CURSIS subroutine.

WINDOW may be used to shrink the effective screen size in order to create protected areas on the screen. The routine may also be used to display multiple plots on the same screen or bring disparate parts of a plot together for comparison.

(a) CALL AXIS

PURPOSE: The AXIS subroutine is used to draw x and y axes, together with tic marks or grid lines, if desired. Each axis may be either linear or logarithmic (base 10).

USAGE: CALL AXIS(XLOW,YLOW,XLNG,YLNG,XTIC,YTIC,MARKX,MARKY)

PARAMETERS:

XLOW - the starting point on the x axis, expressed in scaled, rotated units. XLOW must be the lowest value that the x axis will assume.

If, for example, an x-axis is to be drawn for point a to point b on the figure below,

```
 a------------------------b
 +x ←
```

and if the user has reversed the positive direction of the axis (by calling SCALE with negative XFACT), then b in the figure is in the starting position.

XLNG - the length of the x axis, expressed in scaled units. XLNG is not a signed quantity; AXIS always takes the absolute value of the user's XLNG parameter.

YLNG - the length of the y axis, expressed in scaled units. YLNG is not a signed quantity; AXIS always takes the absolute value of the user's YLNG parameter.

XTIC - the distance between tic marks on the x axis, expressed in scaled units. If XTIC is negative then the x axis will be a log axis, and the absolute value of XTIC will be the distance between decades.
YTIC - the distance between tic marks on the y axis, expressed in scaled units. If YTIC is negative, then the y axis will be a log axis, and the absolute value of YTIC will be the distance between decades.

MARKX - determines axis annotation on the x axis. If MARKX = 0, then no axis annotation will be included. If MARKX > 0, then the x axis will be drawn with tic marks. If MARKX < 0, then the x axis will be drawn with grid lines.

MARKY - determines axis annotation on the y axis. If MARKY = 0, then no axis annotation will be included. If MARKY > 0, then the y axis will be drawn with tic marks. If MARKY < 0, then the y axis will be drawn with grid lines.

DESCRIPTION:

AXIS converts the starting points, XLOW and YLOW, to their raster unit equivalents to determine the correct starting values. XLNG and YLNG are converted to raster units and added to the starting points to determine the correct terminating values. If the user has called ROTATE, then the drawn axis will be appropriately rotated.

AXIS draws a rectangular axis passing through the user's origin and then returns. The routine leaves the terminal in vector mode positioned at the origin. Each axis line specified by the AXIS parameters must pass through the user's origin; if not, the routine returns without drawing an axis.

Example: The user wishes to establish an origin at the centre of the screen. He wishes to draw axes that extend the length of the screen in either direction. Tic marks are to be 20 raster units apart on the x-axis and grid lines are to be 40 raster units apart on the y-axis. The following statements would suffice:

```
CALL SCALE (10.,10.,512.,390.)
CALL AXIS (-51.,-39.,102.,78.,2.,
4.,1,-1)
```

Example: The user wishes to establish an origin at the centre of the screen. He wishes to draw axes that extends 300 raster units in all directions. No tic mark or grid lines are desired, but the axis is to be rotated 45 degrees with respect to the screen margins. The following statements would serve:

```
CALL SCALE (100.,100.,512.,390.)
CALL ROTATE (45.)
CALL AXIS (-3.,-3.,6.,6.,0.,0.,0,0)
```
The plotting routines are used to plot data in the currently selected mode (vectors or points); the mode must be set prior to calling a plotting routine. Thus, a call to TPLLOT while in vector mode will cause a line to be drawn from the current graphics position to the \((x,y)\) point specified as a parameter to TPLLOT. This latter \((x,y)\) then becomes the new current graphics position. In a similar way, a call to TPLLOT while in point plot mode causes a point to be drawn at the \((x,y)\) position specified in the call; this \((x,y)\) point then becomes the new current graphics position.

Two type of plotting routines are provided. TPLLOT takes as parameters an absolute \((x,y)\) point in the user's established coordinate system. DELTA takes as parameters a \((\Delta x,\Delta y)\) relative to the current position. Both TPLLOT and DELTA commence at the current graphics positions, and both update the current graphics position after plotting has occurred.

(a) CALL TPLLOT

PURPOSE: The TPLLOT subroutine is used to plot to an absolute coordinate position in the current mode (vector or point).

USAGE: CALL TPLLOT \( (X,Y, IPEN, MARK) \)

PARAMETERS:

\( X \) - an \( X \) value, expressed in scaled, rotated units.
\( Y \) - a \( Y \) value, expressed in scaled, rotated units.
\( IPEN \) - determines whether the pen is up or down. If \( IPEN = 0 \), then the pen is up and no line is drawn.
If \( IPEN \neq 0 \), then the pen is down and no line is drawn.
\( MARK \) - the number of a data mark taken from the following list:

\( 0 = \text{no data mark} \)
\( 1 = \text{small } x \)
\( 2 = \text{small down arrow} \)
\( 3 = \text{small up arrow} \)
\( 4 = \text{small square} \)
\( 5 = \text{small triangle} \)
\( 6 = \text{small asterisk} \)
\( 11 = \text{large } x \)
\( 12 = \text{large down arrow} \)
\( 13 = \text{large up arrow} \)
\( 14 = \text{large square} \)
\( 15 = \text{large triangle} \)
\( 33-254 = \text{hardware data mark} \)

A hardware data mark is an actual hardware character. If the \( MARK \) parameter is in the range \( (33-254) \), then the ASCII character corresponding to the parameter value is used as the data mark. (ASCII characters are given on p. 13-15).

Any other number other than one of those from the above list is equivalent to no data mark, i.e. \( MARK = 0 \).
DESCRIPTION:

The TPLLOT routine converts the parameters X and Y to raster units and rotates them appropriately prior to plotting. If TPLLOT has been called with an X or Y that, after scaling and rotation, turns out to be outside the current window of vision, then the routine plots only the portion of the plot that appears within the window.

Calling TPLLOT in alpha mode is a legal operation, providing that the pen is up. The effect is to position the alpha cursor and then return the user to alpha mode. However, calling TPLLOT while in alpha mode with the pen down is not functionally defined.

After calling TPLLOT, the current graphics position is appropriately updated.

(b) CALL DELTA

PURPOSE: The DELTA routine is used for local plotting, relative to the current graphics position. Plotting takes place in the current mode (vector or point).

USAGE: CALL DELTA (DELX, DELY, IPEN, MARK)

PARAMETERS:

DELX - a quantity, ΔX, relative to the current X position, expressed in scaled, rotated units.
DELY - a quantity, ΔY, relative to the current Y position, expressed in scaled, rotated units.
IPEN - a pen value that governs whether the pen is up or down. If IPEN = 0, then the pen is up. If IPEN # 0, then the pen is down.
MARK - the number of a data mark. A list of available data marks is given in the description of TPLLOT.

DESCRIPTION:

DELTA converts the parameters DELX and DELY to raster units and adds them to the current graphics position. DELTA then plots to this position in the current mode.

The same conventions that obtain for PLOT also apply to DELTA, viz. plotting is constrained to that portion of the plot within the current window; calling DELTA with pen down while in alpha mode is not functionally defined; and after calling DELTA the current graphics position is appropriately updated.
13.6.5 Cursor Input Routines

The cursor input routines enable the user to input graphic information from the 4010 to his program. When a graphic input routine is called, a full screen crosshair cursor appears on the screen. This cursor may be positioned using the dual thumbwheels on the keyboard. After the cursor is positioned, the user causes position coordinates to be passed back to the graphics input routine by striking a key on the keyboard. When the coordinates have been sent to the computer, the graphics cursor disappears.

Since any key will cause position coordinates of the graphics cursor to be transmitted, the keys may be thought of as special function buttons for graphics input mode. The user might choose one of several subroutines depending upon which key was depressed.

(a) CALL CURRIS

PURPOSE: CURRIS is the basic graphics input routine. When called, CURRIS causes the graphics cursor to appear. The user then positions the cursor and depresses a key.

USAGE: CALL CURRIS(I,X,Y)

PARAMETERS:

I - an ASCII character code corresponding to the key that was depressed.
X - the location of the x-axis corresponding to the cursor position, converted to scaled, rotated units.
Y - the location of the y-axis corresponding to the cursor position, converted to scaled, rotated units.

DESCRIPTION:

CURRIS reads the coordinate position of the cursor and converts it to a scaled coordinate (x,y). These scaled values are then assigned to the user's x and y parameter locations. The ASCII value of the key that was depressed is assigned to a parameter location I. CURRIS may be called while the user is in any mode. After cursor coordinates have been input, the routine will return the user to his current mode.

Calling CURRIS does not affect the user's current graphics position.

(b) CALL SETGRD

PURPOSE: SETGRD is intended for use with the GRID routine. Calling SETGRD partitions the screen into a grid comprised of X*Y rectangles.

USAGE: CALL SETGRD (X,Y, IPEN)

PARAMETERS:

X - the number of regions along the x-axis X ≥ 0
Y - the number of regions along the y-axis Y ≥ 0.
IPEN - If IPEN = 0, then SETGRD values will be retained, but no explicit grid will be drawn. If IPEN ≠ 0, then SETGRD values will be retained and an explicit X×Y grid will be drawn on the screen.

DESCRIPTION:

It is often convenient to draw an explicit grid while debugging a program. When the program is debugged, IPEN can be set to zero and the drawn grid will be suppressed.

SETGRD does some internal rounding in the event that X or Y do not partition the axes uniformly. However, for best results with large X or Y, the user should choose a number of regions that is an integral divisor of the axis in question.

SETGRD may be called from any mode. The user is returned to his current mode by the routine. The parameters X and Y are not affected by the user's current scaling or rotation.

(c) CALL GRID

PURPOSE: GRID is a graphics input routine. It must be used in conjunction with SETGRD. When called, GRID causes the graphics cursor to appear. The user then positions the cursor and depresses a key. GRID returns a row and column coordinate pair that is determined by the cursor location within the grid specified by the last call to SETGRD.

USAGE: CALL GRID (J, JROW, JCOL)

PARAMETERS:

J - an ASCII character code corresponding to the key that was depressed.

JROW - the row of the current grid corresponding to the cursor position.

JCOL - the column of the current grid corresponding to the cursor position.

DESCRIPTION:

GRID allows the user to graphically input a region coordinate in lieu of a point coordinate. It is useful for menu-picking or selecting areas of interest in graphs.

GRID reads the coordinate position of the cursor and converts this value to a (row, column) coordinate based upon the grid size specified in the user's last call to SETGRD. If the cursor falls exactly at the dividing line between two adjacent cells in the grid, the larger coordinate is selected.

Row and column coordinates returned by GRID range from 1 to X and Y respectively. That is, the first row is nominally called row 1; the first column is called column 1. An exception to this indexing scheme occurs if SETGRD was called with X or Y = 0. In this case, GRID will return a zero row or column coordinate corresponding to the axis that had zero partitions. GRID may be called
from any mode. The user is returned to his current mode by this routine.

The parameters JROW and JCOL are not affected by current scaling or rotation.

EXAMPLE: The user wishes to write N lines of descriptive information on the screen. Each line corresponds to a choice. When the program is executed, an option will be selected by positioning the graphics cursor anywhere on the chosen line. Assuming the user has written the code to print N options, one per line, the code to govern menu-picking might look as follows:

C The 4010 has 35 lines in alphanumeric mode.

CALL SETGRD (0,.35,.0)
CALL GRID (KCHAR, LINE, NULL)

C At this point, LINE will have

C a value equal to the index of the chosen option.

13.6.6 Software Character Generation Commands

These commands cause software characters to be drawn out on the screen. WRITEX shifts the current position while WRITEY does not. XLIST is used to convert an integer variable (I to N) or constant to ASCII form so it may be drawn by WRITEX or WRITEY.

(a) CALL WRITEX

PURPOSE: WRITEX is a software character generator. The characters to be written are supplied as parameters to this routine.

USAGE: CALL WRITEX (NUM, JARRAY)

PARAMETERS:

NUM - the number of characters to be generated.
JARRAY - an integer array that contains the ASCII codes of the characters to be generated.

DESCRIPTION:

All characters are drawn in vector mode. The lower left corner of each character commences at the current graphics position. Upon completion, the current position will be updated to the lower left corner of the next character to be drawn.
WRITEX may be called from any mode. The routine will print characters as specified and then return the user to his current mode.

Normal character size is 8x8 raster units; normal print direction is horizontally left to right. WRITEX leaves a 1/4 character width between adjacent characters and a 1/2 character height between adjacent lines. The routine is affected by current scaling and rotation. Thus, it is possible to write smaller, larger, taller, or fatter characters, print at an angle, etc.

The WRITEX character set includes:

- 0 through 9
- A through Z
- space, +, -, , \, /, =

Other characters $, , (, , < have special actions but their use is not recommended. Any character not included in the character set listed above will be ignored.

Characters may be placed into JARRAY using a DATA statement (as in the following example) or by putting the ASCII decimal equivalent of the character into the array (as in the example under WRITEY).

The ASCII decimal equivalent of the characters in the set are given in Table 13.1

<table>
<thead>
<tr>
<th>ASCII</th>
<th>Decimal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>48</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>49</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>51</td>
<td>D</td>
</tr>
<tr>
<td>4</td>
<td>52</td>
<td>E</td>
</tr>
<tr>
<td>5</td>
<td>53</td>
<td>F</td>
</tr>
<tr>
<td>6</td>
<td>54</td>
<td>G</td>
</tr>
<tr>
<td>7</td>
<td>55</td>
<td>H</td>
</tr>
<tr>
<td>8</td>
<td>56</td>
<td>I</td>
</tr>
<tr>
<td>9</td>
<td>57</td>
<td>J</td>
</tr>
<tr>
<td>K</td>
<td>75</td>
<td>V</td>
</tr>
</tbody>
</table>

Table 13.1  ASCII  Decimal Equivalent of Characters.

EXAMPLE: To draw out PAGE 1 GRAPHICS at the top of the screen with triple size characters, using the WRITEX routine. In this case the characters will be placed in JARRAY ( ) using a DATA statement.
C
PROGRAM TO DRAW PAGE 1 GRAPHICS
DIMENSION JARRAY(50)
DATA (JARRAY(I),I=1,10)/2H P,2HAG,2HE ,2H 1,2H ,2H G, 2HRA
12HIC,2HS /
CALL START
CALL ERASE
CALL VECTOR
CALL TFLQT(50.,700.,0,0)
CALL SCALE(3.,3.,0,0)
CALL WRITEX(20,JARRAY)
CALL ALPHA
STOP
END

(b) CALL WRITEY

PURPOSE: WRITEY is a software character generator similar to WRITEX. However, WRITEY allows the user to supply local scaling and rotation parameters.

USAGE: CALL WRITEY (FACT,DEG,NUM,JARRAY)

PARAMETERS:

FACT - a local scaling factor that scales the normal character size.
DEG - a local rotation factor that determines the number of degrees of rotation from the orthogonal system of the screen. Positive rotation is in a counter clockwise direction.
NUM - the number of characters to be generated.
JARRAY - an integer array that contains the ASCII codes of the characters to be generated.

DESCRIPTION: All characters are drawn in vector mode, starting at the current graphics position. Unlike WRITEX, however, WRITEY does not update the current position.

The only scaling factor that affects the character size for WRITEY is FACT. Character size is not influenced by the user's current SCALE.

The only rotation factor that affects WRITEY is DEG. The center of rotation will be the user's current position. This is dissimilar to the convention adopted for WRITEX, where the center of rotation is the user's origin.

WRITEY may be called from any mode. The routine will draw characters as specified and then return the user to his current mode.

The WRITEY character set and character spacing conventions are identical to those established for WRITEX.

EXAMPLE: The user wishes to use WRITEY to draw label TEKPLOT twelve times at different angles around a circle. In this case the characters will be put into JARRAY as ASCII decimal equivalents.
C
PROGRAM TO DRAW ROTATED LABEL.
DIMENSION JARRAY (9)
CALL START
CALL ERASE
CALL TPlot (512.390.0,0)  
centre of screen
JARRAY(1) = 32  
blank
JARRAY(2) = 32  
blank
JARRAY(3) = 84  
T
JARRAY(4) = 69  
E
JARRAY(5) = 75  
K
JARRAY(6) = 80  
P
JARRAY(7) = 76  
L
JARRAY(8) = 79  
O
JARRAY(9) = 84  
T
DO 100 I = 1,331,30  
draw label twelve times
DEG = I - 1
100 CALL WRITEY (2.,DEG,9,JARRAY)  
WRITEY does not update current position
CALL HOME
END

(c) CALL XIMATE

PURPOSE: XIMATE converts the binary representation of a number to the equivalent ASCII representation of that number. The routine can be used to convert an integer variable to a form suitable for use with WRITEX or WRITEY.

USAGE: CALL XIMATE (ICON, NUM, JARRAY)

PARAMETERS:
ICON - an integer variable or constant (<10000) that is to be converted to its ASCII equivalent.
NUM - the number of significant digits, including the sign, in the translated result. NUM will be the number of characters that XIMATE has stored in JARRAY.
JARRAY - an array where XIMATE will store the translated result. JARRAY should be dimensioned large enough to contain one word for each decimal digit in ICON.

DESCRIPTION:
XIMATE suppresses leading zeros. If ICON is positive, the sign position, JARRAY(1), will be an ASCII space; if negative, the sign position will be an ASCII minus.

After calling XIMATE, the assigned NUM and JARRAY parameters may be used as input parameters to WRITEX or WRITEY.
13.7 EXAMPLES

(1) To draw a simple sine curve using TPlot. Note the use of the loop variable to initialize the current graphics location.

C
PROGRAM TO PLOT A SINE CURVE
CALL START
CALL ERASE
CALL VECTOR
CALL SCALE(50.,100.,512.,390.)
DO 100 I=1,631,11
IPEN=I-1
X=I/100
Y=SIN(X)
100 CALL TPlot(X,Y,IPEN,0)
CALL ALPHAN
END

(2) To draw two decades of a log axis using AXIS. Then, for a fixed value of X, the log of points 1 through 9 and 10 through 100 are plotted.

C
PROGRAM TO PLOT LOG AXIS
CALL START
CALL SCALE(1.,250.,512.,100.)
CALL ERASE
CALL AXIS(-500.,0.,1000.,2.,100.,-1.,1,-1.)
CALL POINT
X=500.
DO 100 I=1,10
Y=I
Y=ALOG10(Y)
CALL TPlot(X,Y,1,1)
100 CONTINUE
DO 200 I=10,100,1
Y=I
Y=ALOG10(Y)
CALL TPlot(X,Y,1,1)
200 CONTINUE
CALL ALPHAN
END

(3) To draw an axis; then DELTA is employed to label the axis using hardware (keyboard) character generation. (Note example 4 does equivalent problem by software).

C
PROGRAM TO LABEL AN AXIS BY HARDWARE CHARACTER GENERATION.
CALL START
CALL ERASE
CALL SCALE(10.,10.,200.,200.)
CALL AXIS(0.,0.,50.,5.,1.,1,1)
CALL ALPHAN
CALL DELTA(-1.,-2.,0,0)
DO 100 I = 1,10
CALL DELTA(5.,0.,0,0)
100 WRITE(30,1) I
1 FORMAT(1.,12)
CALL TPlot(0.,0.,0,0)
CALL DELTA(-2.,0.,0,0)

label ticks
write the label. Note that the <or> and <if> do not affect the current graphics location for the next delta
plot back to the origin
DELTA to the west of the axis.
(3) cont.

    DO 200 I=1,5
    CALL DELTA(0.,10.,0.,0.)
200 WRITE(30,2) I
    2 FORMAT(1X,I1)
    CALL ALPHA
    END

(4) To draw the same axis that was specified in Example 3, but the axis labels in this example are generated using the software character generator and XLAB.

C PROGRAM TO LABEL AN AXIS, USING SOFTWARE CHARACTER GENERATION.
   DIMENSION ICHARS(8)
   CALL START
   CALL ERASE
   CALL SCALE(10.,10.,200.,200.) ! origin, scale of 10
   CALL AXIS(0.,0.,50.,5.,10.,-1.,-1.) ! left in VECTOR mode at ORIGIN
C draws a grid axis this time for variety
   CALL ALPHA
   CALL DELTA(-2.5,-5.,0.,0.) ! DELTA to S of axis
   DO 100 I=1,10
   CALL DELTA(5.,0.,0.,0.)
   CALL XLABE(L(I,NUM,ICHARS)) ! translate I to ASCII
100 CALL WRITE(2.,0.,NUM,ICHARS)
   CALL NPLOT(0.,0.,0.,0.)
   CALL DELTA(-4.5,-5.,0.,0.)
   DO 200 I=1,5
   CALL DELTA(0.,10.,0.,0.)
   CALL XLABE(L(I,NUM,ICHARS))
200 CALL WRITE(2.,0.,NUM,ICHARS)
   CALL ALPHA
   END

(5) To show how CURSIS may be used to input graphically sets of (x,y) points (digitizing). Pen values are also input. In this example the points (and the pen values) are both plotted on the screen and saved in a file named USER. Input points are echoed back in VECTOR mode. "Space" echoes as an intensified VECTOR (pen down), while rubout echoes as a dark vector (pen up). DEL ends the program.

C DIGITIZING PROGRAM
   DIMENSION XX(250),YY(250),JIPEN(250),IFCB(14),IBUFF(240)
   DATA IFCB(6),IFCB(9),IFCB(10)/2HUS,2HER,2H /
   CALL START
   CALL ERASE
   CALL VSOPNB(10,15,IFCB,0,11,IBUFF,1) ! save points in file named USER
   JCT=0
   CALL SCALE(1.,1.,512.,380.)
   CALL VECTOR
100 CALL CURSIS(KCHAR,X,Y)
   IF(KCHAR.EQ.7) GO TO 200
   IF(KCHAR.EQ.127) IPEN=0
   CALL NPLOT(X,Y,IFCB,0)
   JCT=JCT+1
   IF(JCT.GT.250) GO TO 200
   XX(JCT)=X
   YY(JCT)=Y
   JIPEN(JCT)=IPEN
GO TO 100
200 DO 300 I=1,JCT
300 WRITE*(10,400) XX(I),YY(I),JIPEN(I) | onto file
400 FORMAT(2(F8.2,2X),I1)
    CALL V$CLSB(10,1)
    CALL HOME
END

(6) This program recalls the (x,y) points and pen values from a file named USER generated in Example 5 and plots these. The program then allows the use of graphics input to specify an area of interest. After defining such an area, the graph is replotted with a new origin and scale factors chosen such that the area of interest exactly fills the whole screen; the remainder of the plot is clipped at the screen margin.

This process demonstrates the use of CURSIS to focus in on selected parts of a complicated plot.

PROGRAM TO READ A SET OF (X,Y) POINTS FROM A FILE
DIMENSION XX(250), YY(250), JIPEN(250) IFCB(14), IBUFF(240)
DATA IFCB(8), IFCB(9), IFCB(10)/2HUS, 2HER, 2H /
    CALL HOME
    CALL V$OPNB(10,15, IFCB, 0,11, IBUFF,1) | open USER
DO 100 I=1,250
100 READ(10,1)X,Y,JIPEN
    FORMAT(2(F8.2,2X),I1)
      IF(IOCHK(J))200,5,200
5   XX(I)=X
   YY(I)=Y
   JIPEN(I)=IPEN
   JCT=I
200 CALL V$CLSB(10,1)
500 XORG=512
     YORG=38C
     XFACT=1.
     YFACT=1.
300 CALL SCALE(XFACT,YFACT,XORG,YORG)
    CALL ERASE
    CALL VECTOR
DO 400 I=1,JCT
400 CALL TPLOT(XX(I),YY(I),JIPEN(I),0)
    CALL CURSIS(K,XMAX,YMAX)
    IF(K.EQ.7) GO TO 90
    IF(K.EQ.82) GO TO 500
    CALL CURSIS(K,XMAX,YMAX)
    IF(K.EQ.82) GO TO 500
    XFACT=1023./ABS(XMAX-XMIN)
    YFACT=780./ABS(YMAX-YMIN)
    XORG=-XMIN*XFACT
    YORG=-YMIN*YFACT
    GO TO 300
90 CALL ALPHA
    STOP
END
ADDENDUM

Since the examples in Appendix I were prepared the line printer has become available.

It can be used for listings, instead of the teletype as shown in the examples. This will give faster operation (200 lpm of 30 cps). In examples 1, 2, and 5 to get a listing on the line printer, the /ASSIGN directive should have LO=LP00 instead of LO=TY00.
APPENDIX 1

EXAMPLES OF PROCEDURE FOR RUNNING FORTRAN PROGRAMMES UNDER VORTEX

1. ENTER PROGRAMME SOURCE FROM PAPER TAPE TO FILE "USER"

/*JOB, JOB1 ▲
   JC**
/ASSIGN LO=TY00 ▼
   JC**
/SEDIT
   SE**
AS,IN=PT
   SE**
AS,OU=15,,USER
   SE**
LI
   SE**
FC
C TEST PROGRAM
   C
     A=10.0
     B=5.0
     C=1.0
     D=A+B+C
     WRITE(14,10)A,B,C,D
10 FORMAT(4G15.7)
STOP
END

/ENDJOB
   JC**

Comments

{ Starts under JCP. /JOB starts this job. Any name may be used for personal identification of output. Return at end of each command. }

{ Call up SEDIT under JCP for entering. SEDIT commands, assigning the SEDIT input and output to the desired logical units - input to PT (to paper tape), output to USER file (scratch file) on disc. }

{ Listing wanted. }

{ File copy - (remains till EOF received). }

Read in program.

Indicate end with EOF (CTRL BEL) on paper tape.

Back to SEDIT.

Back to JCP and end job. (Alternatively can continue with next operation before /ENDJOB.)

NOTES

1. For faster input of tape, the simultaneous listing could be deleted or transferred to the CT¥00. A listing could be obtained by the next program.

2. For the typing in of a program on the keyboard, see the example in the text in Section 4-2.
2. USE "SEDIT" TO LIST A PROGRAMME

/JOB, JOB2
JC**
/ASSIGN, LO=TY00
JC**
/SEDIT

PAGE 1 1

JOB2
VORTEX SEDIT (computer prints information)

SE**
AS,IN=15,,USER
SE**
AS,OU=DU
SE**
LI
SE**
FC
1 C TEST PROGRAM
2 C
3 C
4 A=10.0
5 B=5.0
6 C=1.0
7 D=A+B+C
8 WRITE(1",10)A,B,C,D
9 10 FORMAT(4G15.7)
10 STOP
11 END

SE**
/ENDDJOB
JC**

[ JCP, new job
Assign the list output to TY00 (default is C700) as hard copy of list required.
Call up SEDIT for copying.

SEDIT takes the program in from USER and outputs it to dummy (another copy not needed).

LI is the SEDIT command to list all records copied on to OU logical unit.
File copy.

Listing. Use this to locate errors to be corrected in next use of SEDIT.

[ SEDIT
JCP ]
3. **USE "SEDIT" TO EDIT THE PROGRAMME**

```
/JOB, JOB3
JC**
/SEDIT
SE**
AS, IN=15,,USER
SE**
AS, OU=15,,EDIT
SE**
AS, AL=SI
SE**
DE, 2
SE**
AD, 7
WRITE(14,5)
5 FORMAT(/1 TEST PROGRAM OUTPUT //)
SE**
REPL, 9
10 FORMAT(4G15.5)
SE**
FC
SE**
AS, IN=15,,EDIT
SE**
AS, OU=15,,USER
SE**
FC
SE**
/ENDJOB
JC**
```

- **JOB initiation**
- **Call up SEDIT**
  - **SEDIT commands**
    - IN connected to USER (contains entered program)
    - OU connected to scratch file EDIT on disc
    - Connects alternate input AL to SI logical unit (TTY in this case).
    - Other SEDIT commands
      - DE, 2; delete line 2
      - AD, 7; add after line 7 (input)
      - REPL, 9 replace line 9

- **File copy.**
  - Recopy EDIT back to USER.
  - Connect IN to EDIT, OU to USER and file copy.

- **End job (JCP)**

4. **SAVING THE SOURCE PROGRAMME**

A. **IF NO FILE OF THE NAME CURRENTLY EXISTS.**

```
/JOB, JOB4
JC**
/FMAIN
FM**
CREATE, 16,,TEST1, 120, 10
FM**
/SEDIT
SE**
AS, IN=15,,USER
SE**
AS, OU=15,,TEST1
SE**
FC
SE**
/ENDJOB
JC**
```

- **Initiate job (JCP)**
- **Call up file maintenance software, FMAIN**
  - **FMAIN command to create a file on logical unit number 16, with file name TEST 1 with space for 10 records of 120 words.**
  - **Call up SEDIT**
    - Connect IN to file USER (contains edited program) and OU to file TEST 1 just created.
  - **File copy**
  - **End job JCP**

**NOTE:** With CREATE under /FMAIN, use the rule n = 5 + (no. of lines/3) to give the number of records for storage. Overspecifying n will waste storage space.
B. IF A FILE ALREADY EXISTS

/JOB, JOB4
   JC**
/SEDIT
   SE**
AS,IN=15,,USER
   SE**
AS,OU=16,,TEST1
   SE**
FC
   SE**
/ENDJOB
   JC**

As above without the need to create a new file under FMAIN.

5. COMPILE AND EXECUTE THE PROGRAMME

/JOB, JOB5
   JC**
/ASSIGN, PI=D00K, LO=TY00
   JC**
/PFILE, PI,,USER
   JC**
/MEM,3
   JC**
/FORT,L,M

JCP

Connect PI to disc area D00K which contains the file USER, and LO to teletype.
/PFILE locates the beginning of USER on D00K.
Adds extra 3 lots of ½k core of extra memory for program.
Call up FORTRAN compiler. L indicates outputting binary object on CO file, M suppresses symbol table listing. (If no listing were wanted, add ,M to directive.)

PAGE 1

1 C  TEST PROGRAM
2 C
3   A=1.0
4   B=5.0
5   C=1.0
6   D=A+B+C
7   WRITE(14.5)
8   5 FORMAT(/1 TEST PROGRAM OUTPUT ///)
9   WRITE(14,10)A,B,C,D
10  10 FORMAT(4G15.5)
11   STOP
12   END
Ø ERRORS COMPILATION COMPLETE

computer print out
6. RERUNNING A PROGRAMME SAVED IN A FILE

/JOB, JOB6
JC**
/ASSIGN, PI=16
JC**
/FILE, PI,, TEST1
JC**
/NEM, 3
JC**
/FORT, L, M
JC**
/EXEC

File has been stored on physical device D00J
(see 4). This device has already been
assigned to logical unit no.16, so PI is
now connected to D00J.

Position on beginning of file named TEST1
on D00J.

Extra 3 lots of 1/4 core of memory.
Call FORTRAN compiler.

Execute.

TEST PROGRAM OUTPUT

16.999 5.999 1.9999 51.999 | computer
                print out
SW STOP
JC**
/ENDJOB
JC**

7. EDITING AND RESAVING A SAVED PROGRAMME

/JOB, JOB7
JC**
/SEDIT
SE**
AS, IN=16,, TEST1
SE**
AS,OU=15,, USER
SE**
FC
SE**
/ENDJOB
JC**

Call up SEDIT

Connect IN to TEST1, OU to USER

File copy. Now a copy in USER. Continue
to edit, step 3; and step 4 to save
program.

NOTES: (1) These are only intended as an example of operation and many
equally satisfactory options can be used.

(2) Examples of outputting paper tape or using the line printer
are not shown. It is simply a matter of altering the
assignments to achieve these.

(3) All the examples show an /ENDJOB. If you have several tasks
to perform on the one programme, after completing one it is
possible to reassign the logical units as necessary and continue
with the next task without using /ENDJOB or /JOB.

(4) Be familiar with the options on /FORT. Refer Section 4.2.15 in
the Supplement.

(5) /MEM, n adds n lots of 1/4 of extra core memory. Use only
sufficient for the programme size you have.
LIST OF CHARACTERS  A - Z, ø - ø, $ = + - * / ( ) , . and blank. Other
characters from teletype may be used inside Hollerith (print) fields.

LINE FORMAT  Statement numbers (1 to 99999) in columns 1 - 5.
Continuation (any character except ø) in column 6.
Special characters in column 1; C, * (comment),
X special compile.

VARIABLE NAMES  Up to 6 alphanumeric characters, first is alphabetic
(and $). Classified; integer (implicit I-N), real (other),
double precision, complex, logical.

CONSTANTS  integer (± 32767)
real (± 10^38) must have decimal. E format also.
double precision  D format  e.g. 1.D1ø
complex  ordered pair  e.g. 2., 1.3E4
logical  .TRUE.  or  .FALSE.

OPERATORS  **, /, *, -, +
,.NE. .GE. .GT. .EQ. .LE. .LT. .NOT. .AND. .OR.

SPECIFICATION STATEMENTS

DIMENSION (V1(i 1), V2(i 2),... COMMON/x/ a,b,c
COMMON/ù/ A,R.
EQUIVALENCE (k1), (k2)
EQUIVALENCE (X, A(2),Y)
type a, b, (INTEGER, REAL,
DOUBLE PRECISION, COMPLEX
OR LOGICAL)

STATEMENTS

Var = exp (avoid mixed mode)
GO TO k
GO TO (k1, k2, kn), i
ASSIGN i to m; GO TO m (,,)
IF (exp) k1, k2, k3
IF (exp) stat
CALL s (a1, a2 ...)
RETURN
CONTINUE
PAUSE n
STOP n
DO n i = m1, m2, m3
READ (u,f) k
READ (u) k
WRITE (u,f) k
WRITE (u) k
REWIND u (not relevant)
BACKSPACE u
ENDFILE u

X1 = A* V(I)
GO TO 72
GO TO (98, 4ø5, 3),n
ASSIGN ø to ITEM
IF (I-1) ø,2ø,15
IF (A.LE.1.) GO TO 15
CALL TEST (A,I)

\PAUSE ø777
STOP ø777
DO 15 J = 2, ID,3
READ (13,15) A,B,C
READ (13) (T(J),J=1,N)
WRITE (14,15) (A(I),B(I),I=1,N)
WRITE (14) V7
FORMAT STATEMENTS (see text)

n FORMAT (f1, f2, f3 ... fn) 15 FORMAT (i4)
(fields A, F, E, D, I, X, L, G, T, H, ' '; P scaling)

ROUTINES

ABS ( ), IABS ( ), DABS ( ),
AINT ( ), INT ( ), IDINT ( )
AMOD ( ), MOD ( )
AMAX0 ( ), AMAX1 ( ), MAX0 ( ), MAX2 ( ), DMAX1 ( )
AMIN0 ( ), AMIN1 ( ), MIN0 ( ), MIN1 ( ), DMIN1 ( )
FLOAT ( )
IFIX ( )
SIGN ( ), ISIGN ( ), DSIGN ( )
DIM ( ), IDIM ( ),
SNGL ( )
REAL ( )
AIMAG ( )
DBLL ( )
CMPLX ( )
CONJG ( )
EXP ( ), DEXP ( ), CEXP ( )
 ALOG ( ), DLOG ( ), CLOG ( )
 ALOG10 ( ), DLOG10 ( )
 SIN ( ), DSIN ( ), CSIN ( )
 COS ( ), DCOS ( ), CCOS ( )
 TANH ( )
 SQRT ( ), DSQRT ( ), CSQRT ( )
 ATAN ( ), DATAN ( )
 ATAN2 ( ), DATAN2 ( )
 DMOD ( )
 CABS ( )

SUB-PROGRAMS

FUNCTION f (a1, a2, ...)
SUBROUTINE s (a1, a2, ...)

BLOCK DATA
DATA k1, ... , kn/d1, d2, ...
EXTERNAL s1, s2 ...

FILE ENTRY CALLS

CALL V$OPEN (fun, lun, name, mode) CALL V$OPEN(22,16,IFCB, Ø)
CALL V$CLOS (fun, mode) CALL V$CLOS(22,1)
CALL V$OPNB (fun, lun, name, mode recsvz, buff, rbwfl) CALL V$OPNB(22,15,IFCB, Ø, Ø, JBUFF, 1)
CALL V$CLSB (fun, mode) CALL V$CLSB (22, Ø)
CALL IOCHK (status) CALL IOCHK (ISTAT)
PROCESS INPUT-OUTPUT CALLS

CALL V$OPIO(da,lun,time,stat)   CALL V$OPIO(48,200,40,IST)
CALL AISIQ(W)(count,pplist,ibuf,stat)   CALL AISIQW(1, IPL, IB, IST)
CALL AIRD(W)(count,ptlist,ibuf,stat)   CALL AIRDW(8, IPL, JBF, IST)
CALL AO(W)(count,ptlist,obuf,stat)   CALL AOW(6, IPL, IOB, IST)
CALL DI(W)(count,ptlist,ibuf,stat)   not available
CALL DOM(W)(count,ptlist,obuf,time,stat) CALL DOMW(3, IP, IOB, Ø, IST)
CALL DOL(W)(count,ptlist,obuf,mask,stat) CALL DOLW(3, IL, IOB, IM, IST)
CALL PULSE (chno)   CALL PULSE (5)
CALL STAT (chno,stat)   CALL STAT (6, ISTAT)

GRAPHICS STATEMENTS

CALL START   CALL START
CALL ERASE   CALL ERASE
CALL HOME   CALL HOME
CALL TPAUSE   CALL TPAUSE
CALL ALPHA   CALL ALPHA
CALL VECTOR   CALL VECTOR
CALL POINT   CALL POINT
CALL SCALE (xfact, yfact, xorg, yorg)   CALL SCALE (3,.3,.512,.390.)
CALL ROTATE (deg)   CALL ROTATE (90.)
CALL WINDOW(xmin, xmax, ymin, ymax)   CALL WINDOW (150., 400., 390., 780.)
CALL AXIS (xlow, ylow, xlng, ylng, xtic, ytic, markx, marky)   CALL AXIS (50., 50., 600., 600., 0., 0., 0., 0.0)
CALL TPLT (x, y, ipen, mark)   CALL TPLT (50., 100., 1, 0)
CALL DELTA (delx, dely, ipen, mark)   CALL DELTA (30., 30., 1, 5)
CALL CURSIS (char, x, y)   CALL CURSIS (KHA, XA, VC)
CALL SETGRD (x, y, ipen)   CALL SETGRD (0., 35., 0)
CALL GRID (char, jrow, jcol)   CALL GRID (IKHA, JROW, JCOL)
CALL WRITEX (num, jarray)   CALL WRITEX (20, IARRAY)
CALL WRITEY (fact, deg, num, jarray)   CALL WRITEY (2, 90., Ø, JA)
CALL XLA(T (icon, num, jarray)   CALL XLA(T (1, NUM, ICHAR)
LIST OF FORTRAN IV ERROR MESSAGES

COMPILER

During compilation, source statements are checked for such items as validity, syntax, and usage. When an error is detected, it is posted on the LO usually beneath the source statement. The errors marked T terminate binary output.

All error messages are of the form

`ERR xx c(1)-c(16)`

where `xx` is a number form 0 to 24 (notification error), or T followed by a number from 0 to 32 (terminating error), and `c(1)-c(16)` is the last character string (up to 16) encountered in the statement being processed. The right most character indicates the point of error and the @ indicates the end of the statement. The possible error messages are:

<table>
<thead>
<tr>
<th>Notification Error</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ø</td>
<td>Illegal character input</td>
</tr>
<tr>
<td>1</td>
<td>Construction error</td>
</tr>
<tr>
<td>2</td>
<td>Usage error</td>
</tr>
<tr>
<td>3</td>
<td>Mode error</td>
</tr>
<tr>
<td>4</td>
<td>Illegal DO termination</td>
</tr>
<tr>
<td>5</td>
<td>Improper statement number</td>
</tr>
<tr>
<td>6</td>
<td>Common base lowered</td>
</tr>
<tr>
<td>7</td>
<td>Illegal equivalence group</td>
</tr>
<tr>
<td>8</td>
<td>Reference to nonexecutable statement</td>
</tr>
<tr>
<td>9</td>
<td>No path to this statement</td>
</tr>
<tr>
<td>TØ</td>
<td>Multiply defined statement number</td>
</tr>
<tr>
<td>11</td>
<td>Invalid format construction</td>
</tr>
<tr>
<td>12</td>
<td>Spelling error</td>
</tr>
<tr>
<td>13</td>
<td>Format statement with no statement number</td>
</tr>
<tr>
<td>14</td>
<td>Function not used as variable.</td>
</tr>
<tr>
<td>15</td>
<td>Truncated value</td>
</tr>
<tr>
<td>16</td>
<td>Statement out of order</td>
</tr>
<tr>
<td>17</td>
<td>More than 29 named common regions</td>
</tr>
<tr>
<td>18</td>
<td>Noncommon data</td>
</tr>
<tr>
<td>19</td>
<td>Illegal name</td>
</tr>
<tr>
<td>2Ø</td>
<td>DO index not referenced</td>
</tr>
<tr>
<td>21</td>
<td>Name is dummy</td>
</tr>
<tr>
<td>22</td>
<td>Array name previously declared</td>
</tr>
<tr>
<td>23</td>
<td>Exponent underflow or overflow</td>
</tr>
<tr>
<td>24</td>
<td>Undefined statement number</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Terminating Error</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>TØ</td>
<td>I/O error</td>
</tr>
<tr>
<td>T1</td>
<td>Construction error</td>
</tr>
<tr>
<td>T2</td>
<td>Usage error</td>
</tr>
<tr>
<td>T3</td>
<td>Data pool overflow</td>
</tr>
<tr>
<td>T4</td>
<td>Illegal statement</td>
</tr>
<tr>
<td>T5</td>
<td>Improper use</td>
</tr>
<tr>
<td>T6</td>
<td>Improper statement number</td>
</tr>
<tr>
<td>T7</td>
<td>Mode error</td>
</tr>
<tr>
<td>T8</td>
<td>Constant too large</td>
</tr>
<tr>
<td>T9</td>
<td>Improper DO nesting</td>
</tr>
<tr>
<td>Terminating Error</td>
<td>Definition</td>
</tr>
<tr>
<td>------------------</td>
<td>------------</td>
</tr>
<tr>
<td>T10</td>
<td>DO not parenthesized</td>
</tr>
<tr>
<td>T11</td>
<td>Item not operand</td>
</tr>
<tr>
<td>T12</td>
<td>Item not function</td>
</tr>
<tr>
<td>T13</td>
<td>Invalid unary +, -</td>
</tr>
<tr>
<td>T14</td>
<td>Invalid hierarchy</td>
</tr>
<tr>
<td>T15</td>
<td>Invalid =</td>
</tr>
<tr>
<td>T16</td>
<td>Illegal operator</td>
</tr>
<tr>
<td>T17</td>
<td>Function statement without parameters</td>
</tr>
<tr>
<td>T18</td>
<td>Logical IF follows logical IF</td>
</tr>
<tr>
<td>T19</td>
<td>Invalid dimensions</td>
</tr>
<tr>
<td>T20</td>
<td>Operand is not a name</td>
</tr>
<tr>
<td>T21</td>
<td>Too many numeric characters</td>
</tr>
<tr>
<td>T22</td>
<td>Non-numeric exponent</td>
</tr>
<tr>
<td>T23</td>
<td>Terminator not ,</td>
</tr>
<tr>
<td>T24</td>
<td>Illegal terminator</td>
</tr>
<tr>
<td>T25</td>
<td>Not statement end</td>
</tr>
<tr>
<td>T26</td>
<td>Invalid common type</td>
</tr>
<tr>
<td>T27</td>
<td>Target statement precedes DO</td>
</tr>
<tr>
<td>T28</td>
<td>Subscript variable not dummy</td>
</tr>
<tr>
<td>T29</td>
<td>Title statement not first statement</td>
</tr>
<tr>
<td>T30</td>
<td>First two characters not DO</td>
</tr>
<tr>
<td>T31</td>
<td>Not in subprogram</td>
</tr>
<tr>
<td>T32</td>
<td>Subscript not integer constant</td>
</tr>
</tbody>
</table>

**Note:** Due to optimization the error message may appear on the next labelled statement and not on the actual statement error.

**RUNTIME**

When an error is detected during runtime execution of a program, a message is posted on the LO device of the form

`taskname message`

Fatal errors cause the job to be aborted; execution continues for non-fatal errors. The messages and their definitions are:

<table>
<thead>
<tr>
<th>Message</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARITH OVFL</td>
<td>Arithmetic overflow</td>
</tr>
<tr>
<td>GO TO RANGE</td>
<td>Computer GO TO out of range</td>
</tr>
<tr>
<td>FUNC ARG</td>
<td>Invalid function argument (e.g. square root of negative number)</td>
</tr>
<tr>
<td>FORMAT</td>
<td>Errcr in FORMAT statement*</td>
</tr>
<tr>
<td>MODE</td>
<td>Mode error (e.g. outputting real array with I format)*</td>
</tr>
<tr>
<td>DATA</td>
<td>Invalid input data (e.g. inputting a real number from external medium with I format)*</td>
</tr>
<tr>
<td>I/O</td>
<td>I/O error (e.g. parity, EOF)*</td>
</tr>
</tbody>
</table>

* indicates fatal error; all others non-fatal.
SUMMARY OF COMMANDS FOR JCP, SEDIT, FMAIN, IMGEN, CC

Error messages are given in the Supplement (p 346-358)

A3-1 JOB CONTROL PROCESSOR

/JOB,name
/ENDJOB
/FINT
/C,comment
/MEM,n
/ASSIGN,ll=r1,l2=r2,...

/SREC, lun, nrec, d
/EOF, lun
/RFM, lun, lun,...
/PFILE, lun, key, name
/FORM, lines
/DASMR, p1, p2,...
(BL, LM, N)
/FORT, p1, p2,...
(B, M, L, M, N, 0, X, P)

/CONE
/SEEDIT
/FMAIN
/IMGEN, M
/OUTUL
/SMAIN
/EXEC,D
/LOAD, name, p1, p2,...
/ALTLINE, lun, key
/DUMP

A3-2 SEDIT (Source Editor)

AS, mn = lun, key, file
(nn=IN, OU, AL, default
IN=PI, OU=PO, AL=PI

AD, recno
SA, recno, 'first, last), 'string'
REPL, recno, recno2
SF, recno, (nl, n2, n3), 'string'
DE, recno, recno2
SD, recno, (nl, n2, n3)
MO, recno, recno2, recno3
FC, n (n default=1)
SE, (first, last), init, inc
LT, list (List=A, C, M)
GA, (first, last), 'string'

REPL, p(1), p(2)
CO, (first, last), limit

/JOBM, MAJOR
/EMIN
/FINI (do not use on outside TTY)
/C, FIRST COMPILATION
/NEW, 2
/ASSIGN, PI=CT00, 5=TY

/SREC, PO, 9
/EOF, PO
/RFM, PO, PI
/PFILE, PO, MYFILE
/FORM, 100
/DASMR, H, L
/FORT, L, M

/CONE
/SEEDIT
/FMAIN
/IMGEN, M
/OUTUL
/SMAIN
/EXEC
/LOAD, MYPROG, X1, X2
/ALTLINE, 25, M
/DUMP

AS, IN=1111, , FILFX
AD, 15
SA, 32, (19, 20), ','
REPL, 10, 20
SF, 50, (10, 12, 12), 'XY'
DE, 50, 54
SD, 100, (0, 4, 10)
MO, 5, 10, 99
FC, 16
SE, (15, 17), 30, 7
LT, M
GA, (11, 15), 'VDM'

REPL, IN, ALT, OUT
CO, 5
A3-3  **FMADN (File Maintenance)**

CREATE, lun, key, name, words record
DELETE, lun, key, name
RENAME, lun, key, old, new
ENTER, lun, key, old, new
LIST, lun, key
INIT, lun, key
INPUT, lun, key, file
ADD, lun, key

A3-4  **LGEN (Load Module Generation)**

TTDB, name, type, segments, DEBUG
LD, lun, key, file
OV, signame
LIB, lun1, key1, lun2, key2
END, lun, key

A3-5  **OC OPERATOR COMMUNICATION**

;SCFED, task, level, lun, key
;TSCHED, task, level, lun, key, time
;ATTACH, task, line, ieq, enable
;RESUME, task
;TIME, (time)
;DATE, dcl/mm/yy
;ABORT, task

;SCHED, Y3A, 5, FL, F
;TSCHED, X3A, 5, FL, F, 0930
;ATTACH, X3P, 13, 020
;RESUME, X4C
;TIME
;DATE, 30/07/75
;ABORT, X4C
APPENDIX 4

LIBRARY OF FORTRAN PROGRAMS

The Department is assembling a library of FORTRAN programs, which could be useful to other users. This is still being assembled and a catalogue is not yet available. Further information will be made available at a later date.

If you have fully documented programs which could be of general use to other users, you might consider making it available to the library.

The broad areas of programs covered will be,

(1) General mathematics,
    e.g. matrix operations, soln of e.g. solution of DEs, polynomial approximation

(2) Statistics; regression, analysis of Variance

(3) Optimisation techniques; Fooke & Jeeves, Golden Search

(4) Chemical engineering programs
    e.g. heat exchanger design, column design.

(5) Games

(6) Graphics demonstration programs

(7) Data logging and control examples.

These programs will be kept on cassette and tape, and will be loaded on to disc on request.
GENERAL PROCEDURE FOR PROBLEM SOLVING USING THE DIGITAL COMPUTER

1. DEFINE PROBLEM

Ascerten the purpose of the calculation.
What quantities require to be calculated? Which are provided?
Who will use results? To what purpose? What accuracy is wanted?
How often will program be used? Who will use it?

2. SELECT METHOD OF SOLUTION

How will problem be solved? Direct solution? Trial and error?
Substitution? Numerical simulation?
What simplifying assumptions are made?
Detail a sample calculation (this will be used later in checking).

3. SELECT FORMAT ON INPUT AND OUTPUT

How will data be inputted? within program? while running e.g.
conversational?
How will user know what form of data to put in?
What format of output? Visual impact? What data?

4. FLOW CHART

Draw a flowchart showing the logic of the program steps (e.g.
print headings, input data, perform a calculation, tests, loops,
print results etc.)
The program is prepared from this flowchart.

5. LIST VARIABLE NAMES

Choose variable names to be used in program and list these with
explanation and units. Arranged alphabetically, this list will
be used in the writeup.

6. CODE PROGRAM

Using the appropriate directives encode the steps in the flowchart
into a program. Mark key statement numbers on flowchart as a means
of cross referencing.
Then approach the hardware at last.

7. ENTER PROGRAM INTO COMPUTER

If possible, type program on off line into cassette, tape, cards, etc.
for later rapid inputting to computer.
Correct any typographical or other errors observed at this stage
(EDITING). Input the edited program to the computer.

Alternatively the program can be typed directly into the computer,
but this ties up a lot of facilities. (With a conversational
language, such as PASCAL, there are teaching advantages for on line
typing in for beginners)
8. **EDIT PROGRAM**

Get a printout of the program as stored in the computer and make any (further) corrections as required.

9. **DEBUG PROGRAM**

Run program with known sample values (from ?), find mistakes, and make corrections (debug). Continue with running, correcting and various sample data until the program appears to be working satisfactorily in all its options.

10. **USE PROGRAM**

11. **PRESERVE PROGRAM FOR REUSE**

Obtain permanent reusable record for future use. If use in the near future is intended leave a copy on the bulk storage device within the machine for convenience.

12. **DOCUMENT PROGRAM**

This is important, and unfortunately usually overlooked. It should not be. With the frailty of human memory and the personal nature of programs, to reuse a program which has not been documented is usually a bad investment of time. It is often quicker to rewrite a new one. Therefore documents both within the program by comment statements, and as accompanying document.

The accompanying documents should include,

(i) an unambiguous statement of the purpose of the program and its description.

(ii) a listing of the program

(iii) a detailed flowchart

(iv) full instructions for running (the data required, answers to options, etc.)

(v) a typical set of results.

(vi) a nomenclature list, giving variable names used, their meaning and the chosen units.

(vii) location of copies (reusable) of the program.
PREPARATION OF VARIAN COMPATIBLE TAPE USING PDP-10

If a source program is punched out on paper tape using the PDP-10 it will not be suitable for use on the paper tape reader of the VARIAN. Even though both use ASCII character representation, there is a difference in the use of the eighth hole (parity checking).

A software package is available on the PDP-10 which will produce VARIAN compatible tape. This requires that the program in the PDP-10 be put into a file INPUT. A program IMTAPE is run which creates a file OUTPUT. If this file is punched out it will be compatible with the VARIAN paper tape reader.

Suppose a PDP-10 program is available on paper tape. The steps to produce a paper tape copy suitable for the VARIAN are as follows,

(1) copy the tape into file INPUT by typing

```
  _ RUN TCOPY[136,101]
  ↑ ↑
  (CTRL K) (CTRL M)
  (VT)
  ? R           (for read)
  ? INPUT      (file name)
```

(2) run IMTAPE to give OUTPUT file by typing

```
  _ RUN IMTAPE
```

(3) copy OUTPUT onto tape, by following procedure,

Switch teletype to LOCAL. Prepare leader by typing HEREIS.
Switch off punch
Switch TTY to LINE and type
  COPY TTY:=OUTPUT(I)
Switch punch on immediately to get tape copy.
At end switch TTY to LOCAL for trailing tape.

NOTE: Tape prepared using the teletype off line (LOCAL) is compatible with the VARIAN.

To reduce inefficient usage of the background teletype by on line typing in of programs, it is suggested that all new programs be prepared on paper tape off line. Any available ASR teletype may be used (e.g. PDP-10 terminal, PDP-8 teletype).

Do not use LINEFEED when preparing tape, as this causes problems in the VARIAN.
SECTION 2

REAL-TIME EXECUTIVE SERVICES

The VORTEX real-time executive (RTE) component processes, upon request by a task, operations that the task itself cannot perform, including those involving linkages with other tasks. RTE service requests are made by macro calls to VSEXEC, followed by a parameter list that contains the information required to process the request.

The contents of the volatile A and B registers and the setting of the overflow indicator are saved during execution of any RTE macro. After completion of the macro, these values are returned. The contents of the X register are lost.

There are 32 priority levels in the VORTEX system, numbered 0 through 31. Levels 0 and 1 are for background tasks and levels 2 through 31 are for foreground tasks. If a background task is assigned a foreground priority level, or vice versa, the task automatically receives the lowest valid priority level for the correct environment. Lower numbers assign lower priority.

Background and foreground RTE service requests are similar. However, a level 0 background RTE request causes a memory-protection interrupt and the request is checked for validity. If there is an error, the system prints the error message EX:1 with the name of the task and the location of the violation of memory protection. The background task is aborted.

Table 2.1 | RTE Service Request Macros

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Level 0</th>
<th>FORTRAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHED</td>
<td>Schedule a task</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SUSPND</td>
<td>Suspend a task</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>RESUME</td>
<td>Resume a task</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DELAY</td>
<td>Delay a task</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>LDELAY</td>
<td>Delay and reload from specified logical unit</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>PMSK</td>
<td>Store PIM mask register</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>TIME</td>
<td>Obtain time of day</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>OVLAY</td>
<td>Load and/or execute an overlay segment</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ALOC</td>
<td>Allocate a reentrant stack</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DEALOC</td>
<td>Deallocate the current reentrant stack</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>EXIT</td>
<td>Exit from a task (upon completion)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ABORT</td>
<td>Abort a task</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IOLINK</td>
<td>Link background I/O</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Whenever a task is aborted, all currently active I/O requests are completed. Pending I/O requests are de-queued. Only then is the aborted task released.

There are 12 RTE service request macros. Certain of them are illegal in unprotected background (level 0) tasks. Table 2.1 lists the RTE macros, indicates whether they are legal in level 0 tasks, and indicates whether there is a FORTRAN library subroutine (section 13) provided.

Note: A task name comprises one to six alphanumeric characters (including $), left justified and filled out with blanks. Embedded blanks are not permitted.

2.1 REAL-TIME EXECUTIVE MACROS

This section describes the RTE macros given in Table 2.1.

The general form of an RTE macro is

\[
\text{label} \quad \text{mnemonic}(p(1), p(2), \ldots, p(n))
\]

where:

- \text{label} permits access to the macro from elsewhere in the program
- \text{mnemonic} is one of those given in Table 2.1
- each \(p(n)\) is a parameter defined under the descriptions of the individual macros

The omission of an optional parameter is indicated by retention of the normal number of commas unless the omission occurs at the end of the parameter string. Thus, in the macro (section 2.1.1)

\[
\text{SCHED} \quad 8, , 106, , ' TA', , 'SK', , 'A '
\]

the first double comma indicates a default value for the wait option and the second double comma indicates omission of a protection code.

Error messages applicable to RTE macros are given in Appendix A.2.

2.1.1 SCHED (Schedule) Macro

This macro schedules the specified task to execute on its designated priority level. The scheduling task can pass the
two values in the A and B registers to the scheduled task. The macro has the general form

```
label SCHED level,wait,lun,key,'xx','yy','zz'
```

where

- **level** is the value from 0 (lowest) to 31 (highest) of the priority level of the scheduled task.
- **wait** is 0 (default value) if the scheduling and scheduled task obtain CPU time based on priority levels and I/O activity, or 1 if the scheduling task is suspended until completion of the scheduled task.
- **lun** is the name or number of the logical unit whose library contains the scheduled task, zero to schedule a resident foreground task, or 106 to schedule a nonresident task from the foreground library.
- **key** is the protection code, if any, required to address lun (0306 or 'F' to schedule a nonresident task from the foreground library). The foreground library logical unit and its protection key are specified by the user at system-generation time.
- **xxyyzz** is the name of the scheduled task in six ASCII characters, coded in pairs between single quotation marks and separated by commas; e.g., the task named BIGJOB is coded 'BI', 'GJ', 'OB' and the task named ZAP is coded 'ZA', 'P'.

The FORTRAN calling sequence for this macro is

```
CALL SCHED(level,wait,lun,key,name)
```

where **lun** is the number of the library logical unit containing the task, and **name** is the three-word Hollerith array containing the name of the scheduled task. The other parameters have the definitions given above.

All tasks are activated at their entry-point locations, with the A and B registers containing the values to be passed. The scheduled task executes when it becomes the active task with the highest priority.

The specified logical unit (which can be a background task, a foreground task, or any user-defined library on an RMD) must be defined in the schedule-calling sequence.

**Expansion:** The task name is loaded two characters per word. The wait option flag is bit 12 of word 2 (w).

<table>
<thead>
<tr>
<th>Bit</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 0</td>
<td></td>
<td></td>
<td></td>
<td>J</td>
<td>S R</td>
</tr>
<tr>
<td>Word 1</td>
<td></td>
<td></td>
<td>VSEXEC address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>level</td>
</tr>
<tr>
<td>Word 3</td>
<td></td>
<td></td>
<td>key</td>
<td>lun</td>
<td></td>
</tr>
<tr>
<td>Word 4</td>
<td></td>
<td></td>
<td>Task name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 5</td>
<td></td>
<td></td>
<td>Task name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 6</td>
<td></td>
<td></td>
<td>Task name</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Examples:** Schedule the foreground library task named TSKONE on priority level 5. Use the no-wait option so that scheduled and scheduling tasks obtain Central-Processor-Unit (CPU) time based on priority levels and I/O activity.

```
FL EQU 106 (LUN assigned to foreground library FL)
KEY EQU 0306 (Protection code for FL)
```

```
SCHED 5,0,FL,KEY,'TS','KO','NE'
```

(Control return to highest priority)

**Note:** the KEY line can be coded with the equivalent ASCII character enclosed in single quotation marks.

```
KEY EQU 'F'
```

The same request in FORTRAN is

```
DIMENSION N1(3),N2(3)
DATA N1(1)/2H F/
DATA N2(1),N2(2),N2(3)/2HTS,2HKO,2HNE/
CALL SCHED(5,0,106,N1,N2)
```

or

```
CALL SCHED(5,0,106,2H F,6HTSKONE)
```

### 2.1.2 SUSPND (Suspend) Macro

This macro suspends the execution of the task initiating the macro. The task can be resumed only by an interrupt or a RESUME (section 2.1.4) macro. The macro has the general form

```
label SUSPND susp
```

where susp is 0 if the task is to be resumed by RESUME, or 1 if the task is to be resumed by interrupt.

The FORTRAN calling sequence for this macro is

```
CALL SUSPND(susp)
```

**Expansion:** The susp flag is bit 0 of word 2 (w).
**REAL-TIME EXECUTIVE SERVICES**

Example: Suspend a task from execution. Provide for resumption of the task by interrupt, which reactivates the task at the location following SUSPND.

**SUSPND** 1

The same request in FORTRAN is

CALL SUSPND(1)

### 2.1.3 RESUME Macro

This macro resumes a task suspended by the SUSPND macro. The RESUME macro has the general form

```
label RESUME 'xx','yy','zz'
```

where `xx,yy,zz` is the name of the task being resumed, coded as in the SCHED macro (section 2.1.1).

The RTE searches for the named task and activates it when found. The task will execute when it becomes the task with the highest active priority. If the priority of the specified task is higher than that of the task making the request, the specified task executes before the requesting task and immediately if it has the highest priority.

The FORTRAN calling sequence for this macro is

```
CALL RESUME(name)
```

where `name` is the three-word Hollerith array containing the name of the specified task.

**Expansion:** The task name is loaded two characters per word.

<table>
<thead>
<tr>
<th>Bit</th>
<th>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 0</td>
<td>J S R</td>
</tr>
<tr>
<td>Word 1</td>
<td>VSE EXEC address</td>
</tr>
<tr>
<td>Word 2</td>
<td>0 0 0 0 1 0 0</td>
</tr>
<tr>
<td>Word 3</td>
<td>Task name</td>
</tr>
<tr>
<td>Word 4</td>
<td>Task name</td>
</tr>
<tr>
<td>Word 5</td>
<td>Task name</td>
</tr>
</tbody>
</table>

**Example:** Resume (reactivate) the task TSKTWO, which will execute when it becomes the task with the highest active priority.

```
RESUME 'TS','XT','WO'
```

(Control return)

Control returns to the requesting task when it becomes the task with the highest active priority. Control returns to the location following RESUME.

The same request in FORTRAN is

```
DIMENSION N1(3)
DATA N1(1),N1(2),N1(3)/2HTS,2HTK,2HWO/
CALL RESUME(N1)
```

or

```
CALL RESUME(6HTSKTWO)
```

### 2.1.4 DELAY Macro

This macro suspends the requesting task for the specified time, which is given in two increments. The first increment is the number of 5-millisecond periods, and the second, the number of minutes. The macro has the general form

```
label DELAY milli,min,type
```

where

- `milli` is the number of 5-millisecond increments delay
- `min` is the number of minutes delay
- `type` is 0 (default value) when the task is to be suspended for the specified delay, remain in memory, and automatically resume following the DELAY macro; 1 when the task is to exit from the system, relinquishing memory, and, after the specified delay be automatically rescheduled from the foreground library in a time-of-day mode, or 2 when the task is to resume automatically after the specified delay or upon receipt of an external interrupt, whichever comes first, and automatically resume following the DELAY macro

The FORTRAN calling sequence for this macro is

```
CALL DELAY(milli,min,type)
```

where the integer-mode parameters have the definitions given above.
The maximum value for either milli or min is 32767. Any such combination given the correct sum is a valid delay definition; e.g., for a 90-second delay, the values could be 6000 and 1, respectively, or 18000 and 0. After specified delay, the task becomes active. When it becomes the highest priority active task, it executes.

Note that the resolution of the clock is a user-specified variable having increments of 5 milliseconds. The time interval given in a DELAY macro must be equal to or greater than the resolution of the clock. The delay interval is stored in minute increments and real-time clock resolution increments. Time is kept on a 24-hour clock.

Expansion: The type flag is bits 0 and 1 of word 2.

Example: Assuming a 5-millisecond clock increment, delay the execution of a task for 90 seconds. At the end of this time, the task becomes active. When it becomes the highest priority task, it executes.

**DELAY** 6000, 1

Delay the execution of a task for 90 seconds or until receipt of an external interrupt, whichever comes first, at which time the task becomes active. Such a technique can test devices that expect interrupts within the delay period.

**DELAY** 18000, 0, 2

### 2.1.5 DELAY Macro

This macro is a type 1 DELAY macro with additional parameters to specify the logical unit from which the task is to be reloaded after the delay. The macro has the general form:

```
label DELAY milli,min,lun,key
```

where

- **label** is a label
- **milli** is the number of 5-millisecond increments delay
- **min** is the number of minutes delay
- **lun** is the number of the logical unit from which the task is to be loaded after the delay
- **key** is the protection code for the logical unit

The FORTRAN calling sequence for this macro is

```
CALL LDELAY (milli,min,lun,key)
```

where the integer-mode parameters have the definitions given in the assembly-language form of the call.

Time is the same as specified for DELAY.

### Expansion:

```
Bit 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Word 0                    J S R
Word 1                     VSEXEC address
Word 2 0 0 1 0 0 1         type
Word 3                        milli
Word 4                       min
Word 5                      key    lun
```

Example: Assuming a 5-millisecond clock increment, delay the execution of a task for 90 seconds. At the end of this time, the task becomes active. When it becomes the highest priority task, it is loaded from logical unit 128 which has protection key A, and executed.

```
LDELAY 6000,1,118,0301
```

### 2.1.6 PMSK (PIM Mask) Macro

This macro redefines the PIM (priority interrupt module) interrupt structure, i.e., enables and/or disables PIM interrupts. The macro has the general form

```
label PMSK pim,mask,opt
```

where

- **pim** is the number (1 through 8) of the PIM being modified
- **mask** indicates the changes to the mask, with the bits indicating the interrupt lines that are either to be enabled or disabled, depending on the value of opt, and with the other lines unchanged
- **opt** is 0 (default value) if the set bits in mask indicate newly enabled interrupt lines, or 1 if the set bits in mask indicate newly disabled interrupt lines

The FORTRAN calling sequence for this macro is

```
CALL PMSK(pim,mask,opt)
```

where the integer-mode parameters have the definitions given above.
The eight bits of the mask correspond to the eight priority interrupt lines, with bit 0 corresponding to the highest-priority line.

VORTEX operates with all PIM lines enabled unless altered by a PMSK macro. Normal interrupt-processing allows all interrupts and does one of the following: a) posts (in the TIDB) the interrupt occurrence for later action if it is associated with a lower-priority task, or b) immediately suspends the interrupted task and schedules a new task if the interrupt is associated with a higher-priority task. PMSK provides control over this procedure.

Note: VORTEX (through system generation) initializes all undefined PIM locations to nullify spurious interrupts that may have been inadvertently enabled through the PMSK macro.

Expansion: The opt flag is bit 0 of word 2 (o).

<table>
<thead>
<tr>
<th>Bit</th>
<th>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 0</td>
<td>J S R</td>
</tr>
<tr>
<td>Word 1</td>
<td>V$EXEC address</td>
</tr>
<tr>
<td>Word 2</td>
<td>0 0 1 0 0 0 0</td>
</tr>
<tr>
<td>Word 3</td>
<td>pm mask</td>
</tr>
</tbody>
</table>

Examples: Enable interrupt lines 3, 4, and 5 on PIM 2. Leave all other interrupt lines in the present states.

PMSK 2,070

The same request in FORTRAN is

CALL PMSK(2,56,0)

Disable the same lines.

PMSK 2,070,1

2.1.7 TIME Macro

This macro loads the current time of day in the A and B registers with the B register containing the minute, and the A register the 5-millisecond increments. The macro has the form

`label TIME`

The FORTRAN calling sequence for this macro is

CALL TIME(min,milli)

where min is the hours and minutes in 1-minute integer increments, and milli is the seconds in 5-millisecond integer increments.

Expansion: The opt flag is bit 0 of word 2 (o).

<table>
<thead>
<tr>
<th>Bit</th>
<th>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 0</td>
<td>J S R</td>
</tr>
<tr>
<td>Word 1</td>
<td>V$EXEC address</td>
</tr>
<tr>
<td>Word 2</td>
<td>0 0 1 0 1 0</td>
</tr>
</tbody>
</table>

Example: Load the current time of day in the A (5-millisecond increments) and B (1-minute increments) registers.

TIME

(Return with time in A and B registers)

2.1.8 OVLAY (Overlay) Macro

This macro loads and/or executes overlays within an overlay-structured task. It has the general form

`label OVLAY type,'xx','yy','zz'

where

- **type** is 0 (default value) for load and execute, or 1 for load and return following the request
- **xx** is the name of the overlay segment, coded as in the SCHED macro (section 2.1.1)

The FORTRAN calling sequence for this macro is

CALL OVLAY(type,reload,name)

where type is a constant or name whose value has the definition given above, reload is a constant or name with the value zero to load or non-zero to load only if not currently loaded, and name is a three-word Hollerith array containing the overlay segment name.

FORTRAN overlays must be subroutines if called by a FORTRAN call.

Expansion: The overlay segment name is loaded two characters per word. The type flag is bit 0 of word 2 (t).

<table>
<thead>
<tr>
<th>Bit</th>
<th>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 0</td>
<td>J S R</td>
</tr>
<tr>
<td>Word 1</td>
<td>V$EXEC address</td>
</tr>
<tr>
<td>Word 2</td>
<td>0 0 1 0 1 1</td>
</tr>
<tr>
<td>Word 3</td>
<td>Overlay segment name</td>
</tr>
<tr>
<td>Word 4</td>
<td>Overlay segment name</td>
</tr>
<tr>
<td>Word 5</td>
<td>Overlay segment name</td>
</tr>
</tbody>
</table>
When the load and execute mode is selected in the OVLAY macro RTE executes a JSR instruction to enter the overlay segment. Therefore, the return address of the root segment is available to the overlay segment in the X register.

Example: Find, load, and execute overlay segment OVSG01 without return.

\[
\text{OVLAY } 0, 'OV', 'SG', '01'
\]
(No return)

The same request in FORTRAN is

\[
\text{DIMENSION N1(3)}
\]
\[
\text{DATA N1(1), N1(2), N1(3)/2HOV, 2HSG, 2H01/}
\]
\[
\text{CALL OVLAY(0, 0, N1)}
\]

or

\[
\text{CALL OVLAY(0, 0, 6HOVSG01)}
\]

External subprograms may be referenced by overlays. If a subprogram S is called in several overlays, and S is not in the main segment, each overlay will be built with a separate copy of S.

When using FORTRAN overlays containing I/O statements for RMD files defined by CALL V$OPEN for CALL V$OPNB statements (described in section 5.3.2), the main segment must contain an I/O statement so that the runtime I/O program (V$FORTIO) will be loaded with the main segment. FCB arrays must be in the main segment or in common, so they are linked in memory and cannot be in any overlay.

2.1.9 ALOC (Allocate) Macro

This macro allocates space in a push-down (LIFO) stack of variable length for reentrant subroutines. The macro has the general form

\[
\text{label} \quad \text{ALOC} \quad \text{address}
\]

where address is the address of the reentrant subroutine to be executed.

The FORTRAN calling sequence for this macro is

\[
\text{EXTERNAL ALOC(subr)}
\]

where subr is the name of the DAS MR assembly language subroutine.

The first location of the LIFO stack is V$LOC, and that of the current position in the stack is V$CRS. The first word of the reentrant subroutine, whose address is specified in the general form of ALOC, contains the number of words to be allocated. If fewer than five words are specified, five words are allocated.

Control returns to the location following ALOC when a DEALOC macro (section 2.1.8) is executed in the called subroutine. Between ALOC and DEALOC, (1) subroutine cannot be suspended, (2) no LOC calls (section 3) can be made, and (3) no RTE service calls can be made.

Reentrant subroutines are normally included in the resident library at system-generation time so they can be concurrently accessed by more than one task. The maximum size of the push-down stack is also defined at system-generation time.

Expansion:

<table>
<thead>
<tr>
<th>Bit</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 0</td>
<td>J</td>
<td>S</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 1</td>
<td>V$EXEC address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 3</td>
<td>Reentrant subroutine address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reentrant subroutine: The reentrant subroutine called by ALOC contains, in entry location x, the number of words to be allocated. Execution begins at x + 1. The reentrant subroutine returns control to the calling task by use of a DEALOC macro.

The reentrant stack is used to store register contents and allocate temporary storage needed by the subroutine being called. The location V$CRS contains a pointer to word 0 of the current allocation in the stack. By loading the value of the pointer into the X (or B) register, temporary storage cells can be referenced by an assembly language M field of 5,1 for the first cell; 6,1 for the second; etc.

A stack allocation generated by the ALOC macro has the format:

\[
\text{Bit} \quad 15 \quad 14 \quad 13 \quad 12 \quad 11 \quad 10 \quad 9 \quad 8 \quad 7 \quad 6 \quad 5 \quad 4 \quad 3 \quad 2 \quad 1 \quad 0
\]

| Word 0 | Contents of the A register |
| Word 1 | Contents of the B register |
| Word 2 | Contents of the X register |
| Word 3 | ovfl |
| Word 4 | Stack control pointer (for RTE use only) |
| Word 5 | For reentrant subroutine use (temporary storage) |
| ... | ... |
| Word n | |

where ovfl is the overflow indicator bit.
The current contents of the A and B registers are stored in words 0 and 1 of the stack and are restored upon execution of the DEALOC macro. The same procedure is used with the setting of the overflow indicator bit in word 3 of the stack. The contents of word 2 (X register) point to the location of the reentrant subroutine to be executed following the setting up of the stack. The contents of word 3 (bits 14-0) point to the return location following ALOC.

Example: Allocate a stack of six words. Provide for deallocation and returning of control to the location following ALOC:

```
  EXT SUB1
  ALOC SUB1
      (Return Control)
  ...
  NAME SUB1
  SUB1 DATA 6
      ...
  DEALOC
  END
```

Each time SUB1 is called, six words are reserved in the reentrant stack. Each time the reentrant subroutine makes a DEALOC request (section 2.1.8), six words are deallocated from the reentrant stack.

### 2.1.10 DEALOC (Dealocate) Macro

This macro deallocates the current reentrant stack, restores the contents of the A and B registers and the setting of the overflow indicator to the requesting task, and returns control to the location specified in word 3 (P register value) of the reentrant stack (section 2.1.7). The macro has the form

```
label DEALOC
```

**Expansion:**

```
Bit  15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Word 0  J S R
Word 1  V$EXEC address
Word 2  0 0 0 1 1 1
```

Example: Release the current reentrant stack, restore the contents of the volatile registers and the setting of the overflow indicator and return control to the location specified in word 3 of the stack.

```
  ...
  DEALOC
  END
```

### 2.1.11 EXIT Macro

This macro is used by a task to signal completion of that task. The requesting task is terminated upon completion of its I/O. The macro has the form

```
label EXIT
```

The FORTRAN calling sequence (no parameters specified) is

```
CALL EXIT
```

If the task making the EXIT is in unprotected background memory, the macro schedules the job-control processor (JCP) task (section 4).

**Expansion:**

```
Bit  15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Word 0  J S R
Word 1  V$EXEC address
Word 2  0 0 0 0 0 1 0
```

Example: Exit from a task. The task making the EXIT call is terminated upon completion of its I/O requests.

```
  ...
  EXIT   (No return)
```

### 2.1.12 ABORT Macro

This macro aborts a task. Active I/O requests are completed, but pending I/O requests are dequeued. The macro has the general form

```
label ABORT 'xx','yy','zz'
```

where xxx yyy zzz is the name of the task being aborted, coded as in the SCHED macro (section 2.1.1).

The FORTRAN calling sequence for this macro is

```
CALL ABORT(name)
```

where name is the three-word Hollerith array containing the name of the task being aborted.

**Expansion:** The task name is loaded two characters per word.

```
Bit  15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Word 0  J S R
Word 1  V$EXEC address
Word 2  0 0 0 0 1 0 1
Word 3  Task name
Word 4  Task name
Word 5  Task name
```
Example: Abort the task TSK and return control to the location following ABORT.

*  
*  
ABORT 'TS', 'K', ' ' 
  (Control return)  
*  
The same request in FORTRAN is

DIMENSION N1(3)  
DATA  N1(1), N1(2), N1(3)/2HTS, 2HK, 2H /  
CALL ABORT(N1)

or

CALL ABORT(6HTSK )

2.1.13 IOLINK (I/O Linkage) Macro

This macro enables background tasks to pass buffer address and buffer size parameters to the system background global FCBS. It has the general form

```
label: IOLINK lungsd, bufloc, bufsiz
```

where

- `lungsd` is the logical unit number of the global system device
- `bufloc` is the address of the input/output buffer
- `bufsiz` is the size of the buffer (maximum and default value: 120)

Global file control blocks: There are eight global FCBS (section 3.5.11) in the VORTEX system reserved for background use. System background and user programs can reference these global FCBS. JCP directive /PFILE (section 4.2.12) stores the protection code and file name in the corresponding FCB before opening/rewinding the logical unit. The IOLINK service request passes the buffer address and the size of the record to the corresponding logical-unit FCB. The names of the global FCBS are SIFCB, PIIFCB, POFCB, SSFCB, BIFCB, BOFCB, GOFCB, and LOFCB, where the first two letters of the name indicate the logical unit.

Expansion:

<table>
<thead>
<tr>
<th>Bit</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 0</td>
<td>R</td>
<td>S</td>
<td>J</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 1</td>
<td>VSEXEC address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>lungsd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 3</td>
<td>bufloc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 4</td>
<td>bufsiz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example: Pass the address and size specifications of a 40-word buffer at address BUF to the PI global FCB.

```
PI EQU 4
EXT PIIFCB  
  (PI logical-unit number 4)  
  
IOLINK PI, BUF, 40

READ PIIFCB, P1, 0, 1  
  (Read 40 ASCII words  
  from PI)  
  
BUF BSS 40
END
```

If the PI file is on an RMD, reassign the PI to the proper RMD partition, and then position the PI file using JCP directive /PFILE.

2.1.14 TBEVNT (Set or Fetch TBEVNT) Macro

This macro fetches or sets the requesting task's event word, TBEVNT, word 3 of the TIDB. It can also be used to change other words in the TIDB. However, most changes to entries in the TIDB could cause irrecoverable errors, so the TBEVNT macro should be used only with caution. Section 14 gives information about the format and contents of the TIDB.

This macro has the general form

```
label: TBEVNT value, disp, c/s
```

where

- `value` is a value or bit mask for the specified TIDB word. If disp is 0, value 0·01777776 changes the TBEVNT word and a value of 01777777 fetches the TBEVNT contents into the A register. If disp is not zero, it sets or resets (depending on c/s) the word specified by disp
- `disp` is the displacement of the word in the TIDB to be set or reset, or 0 for TBEVNT (word 3). The default is 0
- `c/s` is the clear or set indicator, if disp is not 0. c/s = 0 for clear (the zero bits of the value indicate the bits of the specified word to be set to 0) and 1 for clear (the one bits in value indicate the bits to be set to 1). Default is 0
### Bit Table

<table>
<thead>
<tr>
<th>Bit</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 0</td>
<td></td>
<td>J</td>
<td>S</td>
<td>R</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 1</td>
<td>disp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 2</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 3</td>
<td>Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 4</td>
<td>disp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Default values: disp = 0, c/s = 0

**Example:** Save the value of TBEVNT in TEMP then set TBEVNT to 02.

```plaintext
START       TBEVNT  0177777
            STA      TEMP  (Save TBEVNT)
            TBEVNT  02  (Set TBEVNT = 2)
            .
            .
 TEMP       BSS     1
```

---

### Example

Reset TBPL (word 2 of TIDB) bit 8 and then set it again.

- **TBEVNT**: 0177777, 2, 0
- **TBEVNT**: 0400, 2, 1

---

#### 2.2 ABORT PROCEDURE

Whenever a task is aborted, all currently active I/O operations are allowed to complete. All I/O requests that are threaded (queued, or waiting to be activated) are not activated. Upon completion of all active I/O operations and after all pending requests are de-threaded, the aborted task is released.
SECTION 4
JOB-CONTROL PROCESSOR

The job-control processor (JCP) is a background task that permits the scheduling of VORTEX system or user tasks for background execution. The JCP also positions devices to required files, and makes logical-unit and I/O-device assignments.

4.1 ORGANIZATION

The JCP is scheduled for execution whenever an unsolicited operator key-in request (section 17.2) to the OC logical unit has a slash (/) as the first character.

Once initiated, the JCP processes all further JCP directives from the SI logical unit.

If the SI logical unit is a Teletype or a CRT device, the message JCP** is output to indicate the SI unit is waiting for JCP input. The operator is prompted every 15 seconds (by a bell for the Teletype or tone for the CRT) until an input is keyed in.

If the SI logical unit is a rotating-memory-device (RMD) partition, the job stream is assumed to comprise unblocked data. In this case, processing the job stream requires an /ASSIGN directive (section 4.2.6).

A JCP directive has a maximum of 80 characters, beginning with a slash. Directives input on the Teletype are terminated by the carriage return.

4.2 JOB-CONTROL PROCESSOR DIRECTIVES

This section describes the JCP directives:

a. Job-initiation/termination directives:

/JOB Start new job
/ENDJOB Terminate job in progress
/FINI Terminate JCP operation
/C Comment
/MEM Allocate extra memory for background task

b. I/O-device assignment and control directives:

/ASSIGN Make logical-unit assignment(s)
/SFILE Skip file(s) on magnetic-tape unit
/SREC Skip record(s) on magnetic-tape unit or RMD partition
/WEOF Write end-of-file mark
/REW Rewind magnetic-tape unit or RMD partition
/PFILE Position rotating memory-unit file
/FORM Set line count on LO logical unit
/KP/MODE Set keypunch mode
/OPEN Open VTAM line or terminal
/CLOSE Close VTAM line

- Language-Processor directives:
  /DASMR Schedule DAS MR assembler
  /FORT Schedule FORTRAN compiler

d. Utility directives:

/CONC Schedule system-concordance program
/SEDIT Schedule symbolic source-editor task
/FMAIN Schedule file-maintenance task
/LMGEN Schedule load-module generator
/IOUTIL Schedule I/O-utility processor
/SMAIN Schedule system-maintenance task

e. Program-loading directives:

/EXEC Schedule loading and execution of a load-module from the SW unit file
/LOAD Schedule loading and execution of a user background task
/ALTLIB Schedule the next background task from the specified logical unit rather than from the background library
/DUMP Dump background at completion of task execution

JCP directives begin in column 1 and comprise sequences of character strings having no embedded blanks. The character strings are separated by commas (,) or by equal signs (=). The directives are free-form and blanks are permitted between the individual character strings of the directive, i.e., before or after commas (or equal signs). Although not required, a period (.) is a line terminator. Comments can be inserted after a period.

Each JCP directive begins with a slash (/).

The general form of a job-control statement s

/s, p(1), p(2), ..., p(n)

where

name is one of the directive names given (any other character string produces an error)

each p(n) is a parameter required by the JCP or by the scheduled task and defined below under the descriptions of the individual directives

Numerical data can be octal or decimal. Each octal number has a leading zero.

* DO NOT USE, unless have access to OC device.
For greater clarity in the descriptions of some directives, optional periods, optional blank separators between character strings, and the optional replacement of commas by equal signs are omitted from descriptions.

Error messages applicable to JCP directives are given Appendix A.4.

4.2.1 /JOB Directive

This directive initializes all background system pointers and flags, and stores the job name if one is specified. It has the general form

/JOB, name

where name is the name of the job and comprises up to eight ASCII characters (additional characters are permitted but ignored by the JCP).

The job name, if any, is then printed at the top of each page for all Vortex background programs.

Example: Initialize the job TASKONE.

/JOB, TASKONE

4.2.2 /ENDJOB Directive

This directive initializes all background system pointers and flags, and dews the job name. It has the form

/ENDJOB

Example: Terminate the job in process.

/ENDJOB

* 4.2.3 /FINI (Finish) Directive

This directive terminates all JCP background operations and makes an EXIT request to the real-time executive RTE component (section 2.1.11). It has the form

/FINI

To reschedule JCP after a FINI, input any JCP directive from the OC unit (section 17).

Example: Terminate JCP operations.

/FINI

* DO NOT USE, unless have access to OC device.

4.2.4 /C (Comment) Directive

This directive outputs the specified comment to the SO and LO logical units, thus permitting annotation of the listing. It is not otherwise processed. It has the general form

/C, comment

where comment is any desired free-form comment.

Example: Annotate a listing with the comment Rewind all mag tapes.

/C, REWIND ALL MAG TAPES

4.2.5 /MEM (Memory) Directive

This directive assigns additional 512-word blocks of main memory to the next scheduled background task. It has the general form

/MEM, n

where n is the number of 512-word blocks of main memory to be assigned.

/MEM permits larger symbol tables for FORTRAN compilations and DAS MR assemblies.

The total area of the 512-word blocks of memory plus the background program itself cannot be greater than the total area available for background and nonresident foreground tasks. An attempt to exceed this limit causes the scheduled task to be aborted.

Example: Allocate an additional 1,024 words of main memory to the next scheduled task.

/MEM, 2

4.2.6 /ASSIGN Directive

This directive equates and assigns particular logical units to specific I/O devices. It has the general form

/ASSIGN, (l(1)) = r(1), (l(2)) = r(2), ..., (l(n)) = r(n)

where

each l(n) is a logical-unit number (e.g., 102) or name (e.g., SI)

each r(n) is a logical-unit number or name, or a physical-device system name (e.g., TY00, table 15-1)

The logical unit to the left of the equal sign in each pair is assigned to the unit/device to the right.
If the controller and unit numbers are omitted from the name of a physical device, controller 0 and unit 0 are assumed.

An inoperable device, i.e., one declared down by the :DEVDN operator key in request (section 17.2.10), cannot be assigned. A logical unit designated as unassignable cannot be reassigned.

Example: Assign the PI logical unit to card reader CR00 and the LO logical unit to Teletype TY00.

/ASSIGN, PI=CR, LO=TY

4.2.7 /SFILE (Skip File) Directive

This directive, which applies only to magnetic-tape units and card readers, causes the specified logical unit to move the tape forward the designated number of end-of-file marks. It has the general form

/SFILE, lun, neof

where

lun is the number or name of the affected logical unit
neof is the number of end-of-file marks to be skipped

If the end-of-tape mark is encountered before the required number of files has been skipped, the JCP outputs to the SO and LO logical units the error message JC05,nn, where nn is the number of files remaining to be skipped.

Example: Skip three files on the BI logical unit.

/SFILE, BI, 3

4.2.8 /SREC (Skip Record) Directive

This directive, which applies only to magnetic-tape units, card readers, and RMDs, causes the specified logical unit to move the tape the designated number of records in the required direction. In the case of RMDs, word 4 of the FCB is adjusted the appropriate number of records. It has the general form

/SREC, lun, nrec, direc

where

lun is the number or name of the affected logical unit
nrec is the number of records to be skipped
direc indicates the direction to be skipped; F (default value) for forward, or R for reverse. Reverse skip does not apply to the card reader.

If a file mark, end of tape, or beginning of tape is encountered before the required number of records has been skipped, the JCP outputs to the SO and LO logical units the error message JC05,nn, where nn is the number of records remaining to be skipped.

Example: Skip nine records forward on the BO logical unit.

/SREC, BO, 9

4.2.9 /WEOF (Write End of File) Directive

This directive writes an end-of-file mark on the specified logical unit. It has the general form

/WEOF, lun

where lun is the number or name of the affected logical unit.

Example: Write an end-of-file mark on the BO logical unit.

/WEOF, BO

4.2.10 /REW (Rewind) Directive

This directive, which applies only to magnetic-tape units and RMDs, causes the specified logical unit(s) to rewind to the beginning of tape. It has the general form

/REW, lun, lun, ..., lun

where lun is the number or name of a logical unit to be rewound.

Example: Rewind the BO and PI logical units.

/REW, BO, PI

4.2.11 /PEFILE (Position File) Directive

This directive, which applies only to RMDs, causes the specified logical unit to move to the beginning of the designated file. It has the general form
JOB-CONTROL PROCESSOR

/PFILE, lun, key, name

where

lun is the number or name of the affected
logical unit. The logical unit must be one
of the system defined logical units which
has a global FCB

key is the protection code required to
address lun

name is the name of the file to which the
logical unit is to be positioned

Global file control blocks: There are eight global file
control blocks (FCB, section 3.5.11) in the VORTEX system
that are reserved for background use. System background
and user programs can reference these global FCBs. The
/PFILE directive stores key and name in the corresponding
FCB before opening/rewinding the logical unit. To pass
the buffer address and size of the record to the corresponding
logical-unit FCB, make an RTE IOLINK service request
(section 2.1.12). The names of the global FCBs are SIFCB,
PIFCB, POFCB, SSFCB, BIFCB, BOFCB, GOFCB, and
LOFCB, where the first two letters of the name indicate the
logical unit.

Example: Position the PI logical unit to beginning of file
FILEXY, whose protection key is $.

/PFILE, PI, $, FILEXY

4.2.14 /DASMR (DAS MR Assembler)
Directive

This directive schedules the DAS MR assembler (section
5.1) with the specified options for background operation on
priority level 1. It has the general form

/DASMR, p(1), p(2), ... p(r)

where each p(n), if any, is a single character specifying one
of the following options:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Presence</th>
<th>Absence</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Suppresses binary object</td>
<td>Output binary object</td>
</tr>
<tr>
<td>L</td>
<td>Outputs binary object on GO file</td>
<td>Suppresses output of binary object on GO file</td>
</tr>
<tr>
<td>M</td>
<td>Suppresses symbol-table listing</td>
<td>Output symbol-table listing</td>
</tr>
<tr>
<td>N</td>
<td>Suppresses source listing</td>
<td>Outputs source listing</td>
</tr>
</tbody>
</table>

The /DASMR directive can contain up to four such
parameters in any order.

The DAS MR assembler reads source records from the PI
logical unit on the first pass. The PI unit must have been
set to the beginning of device before the /DASMR directive.
This can be done with an /ASSGN (section 4.2.6), /SFILE
(section 4.2.7), /REW (section 4.2.10), or /PFILE (section
4.2.11) directive.

A load-and-go operation requires, in addition, an /EXEC
directive (section 4.2.22).

Example: Schedule the DAS MR assembler with no source
listing, but with binary-object output on the GO file.

/JOB, EXAMPLE /PFILE, BO, BO /DASMR, N, L

/ JOB initializes the GO file to start of file. If BO is assigned
to a rotating memory partition, a /PFILE, BO, BO must pre-
cede the /DASMR directive to initialize the file (unless the
assembly is part of a stacked job - see section 4.3 for sample
deck setup).

4.2.14 /KPMODE (Keypunch mode)
Directive

This directive specifies the mode, 026 or 029, (BCD or
EBCDIC respectively) in which VORTEX is to read and
punch cards. It has the general form

/KPMODE, m

where m is 0 (default value) for 026 mode, or 1 for 029
mode.

Example: Specify that cards be read and punched in 029
keypunch mode.

/KPMODE, 1
4.2.15 /FORT (FORTRAN Compiler)

Directive

This directive schedules the FORTRAN compiler (section 5.3) with the specified options for background operation on priority level 1. It has the general form

/FORT,p(1),p(2),...,p(n)

where each p(n), if any, is a single character specifying one of the following options:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Presence</th>
<th>Absence</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Suppresses binary object</td>
<td>Output binary object</td>
</tr>
<tr>
<td>D</td>
<td>Assigns two words to integer array items and to integer and logical variables (ANSI standard)</td>
<td>Assigns one word to integer array items and to integer and logical variables</td>
</tr>
<tr>
<td>H</td>
<td>Generate code using Floating Point Processor (FPP)</td>
<td>Generate no FPP instructions</td>
</tr>
<tr>
<td>L</td>
<td>Outputs binary object on GO file</td>
<td>Suppresses output of binary object on GO file</td>
</tr>
<tr>
<td>M</td>
<td>Suppresses symbol-table listing</td>
<td>Outputs symbol-table listing</td>
</tr>
<tr>
<td>N</td>
<td>Suppresses source listing</td>
<td>Outputs source listing</td>
</tr>
<tr>
<td>O</td>
<td>Outputs object-module listing</td>
<td>Suppresses object-module listing</td>
</tr>
<tr>
<td>X</td>
<td>Compiles conditionally</td>
<td>Compiles normally</td>
</tr>
<tr>
<td>F</td>
<td>Generates code with calls to faster firmware routines (see section 20.2)</td>
<td>Generates subroutine calls</td>
</tr>
</tbody>
</table>

The /FORT directive can contain up to 7 such parameters in any order.

A load-and-go operation requires, in addition, an /EXEC directive (section 4.2.22).

Example: Schedule the FORTRAN compiler with binary-object, source, symbol-table, and object-module listings; normal compilation; and no binary-object output on the GO file.

/FORT,O

4.2.16 /CONC (System Concordance)

Directive

This directive schedules the system concordance program (section 5.2) for background operation. It has the form

/CONC

The concordance program inputs from the SS logical unit and uses the same source statements that are input to the DAS MR assembler. It outputs to the LO logical unit a listing of all symbols and their referenced locations in the same input program.

The SS unit is set to the beginning of device before the /CONC directive.

Example: Schedule the system concordance program.

/ASSIGN,SS=MTO0
/REW,SS
/DASMR
/PFILE,SS,SS
/CONC

4.2.17 /SEDIT (Source Editor)

Directive

This directive schedules the symbolic source editor (section 8) for background operation on priority level 1. It has the form

/SEDIT

Example: Schedule the symbolic source editor.

/SEDIT

4.2.18 /FMAIN (File Maintenance)

Directive

This directive schedules the file maintenance task (section 9) for background operation on priority level 1. It has the form

/SEDIT
4.2.19 /LMGEN (Load-Module Generator) Directive

This directive schedules the load-module generator (section 6) for background operation on priority level 1. A memory map is output unless suppressed. The directive has the general form

/LMGEN,M

where M, if present, suppresses the output of a memory map.

Example: Schedule the load-module generator task without a memory map.

/LMGEN,M

4.2.20 /IOUTIL (I/O Utility) Directive

This directive schedules the I/O utility processor (section 10) for background operation on priority level 0. The directive has the form

/IOUTIL

Example: Schedule the I/O utility processor.

/IOUTIL

4.2.21 /SMAIN (System Maintenance) Directive

This directive schedules the system maintenance task (section 16) for background operation on priority level 1. The directive has the form

/SMAIN

Example: Schedule the system maintenance task.

/SMAIN

4.2.22 /EXEC (Execute) Directive

This directive schedules the load-module loader to load and execute a load module from the SW logical unit file. Since this is not a VORTEX system task, execution is on priority level 0. The directive has the general form

/EXEC,D

Where D, if present, dumps all of the background upon completion of execution. The dump format consists of eight memory locations per line. Both octal and ASCII representation appear in the dump. During ASCII dump non-ASCII characters appear as blanks. ASCII dump is suppressed if dump is to a TY or CT device.

The dump format consists of eight memory locations per line as follows:

XXXXXXXX AAAAAA BBBBBB... HHHHHH

where XXXXXX is the starting memory address location of the eight following data words and AAAAAA through HHHHHH are the octal values of those locations. The occurrence of an asterisk between two lines indicates that all dump lines between those lines have the same value as the previous line.

Example: Schedule the loading of a user load module from the SW unit file without a background dump.

/EXEC

4.2.23 /LOAD Directive

This directive schedules a user task, which must be present in the background library or alternate library, for background execution on priority level 0. The directive has the general form
4.2.25 /DUMP Directive

This directive causes all of background to be dumped upon completion of execution of a task executed from the background library or an alternate library. The dump format is the same as the format for /EXEC.D (see section 4.2.22).

Example: Schedule the execution of user task TSKONE with a dump at completion of execution.

/DUMP
/LOAD, TSKONE

4.3 SAMPLE DECK SETUPS

The batch-processing facilities of VORTEX are invoked by JCP control directives in combination with programs and data. These elements form the input job stream to VORTEX. The input job stream can come from various peripherals and be carried on various media. These examples illustrate common job streams and deck-preparation techniques.

Example 1 - Card Input: Compile a FORTRAN IV main program (with source listing and octal object listing), and assemble a DAS MR subprogram. Then load and execute the linked program.

/JOB, EXAMPLE1
/FORT, L, 0

(Source Deck)

/DASMR, L

(Source Deck)

/EXEC
/ENDJOB

Example 2 - Card Input: Assemble a DAS MR program (with source listing and load-and-execute) and generate a concordance listing. The DAS MR program is cataloged on RMD partition D00K under file name USER1 with protection key U. Assign the PI logical unit to RMD partition D00K, open file name USER1 for the assembler, assemble the program, and execute the program with a dump.

/JOB, EXAMPLE2
/ASSIGN, PI=D00K
/PFILE, PI, U, USER1
/DASMR, L
/PFILE, SS, SS
/CONC
/EXEC, D
/ENDJOB
Example 3 - Card Input: Assemble a DAS MR program (with source listing and object-module output on the BO logical unit). Assign the PI logical unit to magnetic-tape unit MT00, the PO logical unit to dummy device, the SS logical unit to the PI logical unit, the BO logical unit to RMD partition D00J, and output the object module to file name USER2 with no protection key. Before assembly, position the PI logical unit to the third file. Allocate four additional 512-word blocks for the DAS MR symbol-table area.

/JOB,EXAMPLE3
/ASSIGN,P=MT00,PO=DUM,SS=PI,BO=D00J
/REW,PI
/SFILE,PI,2
/PFILE,BO,USER2
/MEM,4
/DASMR
/ENDJOB

Example 4 - Card Input: After generation of a VORTEX system, use FMAIN to initialize and add object modules to the object-module library (OM) with protection key D. Assign the BI logical unit to CR00.

/JOB,EXAMPLE4
/ASSIGN,BI=CR00
/FMAIN
/EXEC
INIT,OM,D
INPUT,BI
ADD,OM,D

Example 5 - Card Input: Load and go operation. Compile a FORTRAN IV main program, a subprogram and assemble a DASMR subprogram. Save output on BO. Execute the linked programs.

/JOB,EXAMPLE5
/PFILE,BO,BO
/FORT,L
/EXEC

(Dasmr,L

(Source deck FORTRAN main program)

(Source deck FORTRAN subprogram)

/EXEC
/FINI
SECTION 6 
LOAD-MODULE GENERATOR

The load-module generator (LMGEN) is a background task that generates background and foreground tasks from relocatable object modules. The tasks can be generated with or without overlays, and are in a form called load modules.

To be scheduled for execution within the VORTEX operating system, all tasks must be generated as load modules.

6.1 ORGANIZATION

LMGEN is scheduled for execution by inputting the job-control processor (JCP) directive /LMGEN (section 4.2.19).

LMGEN has a symbol-table area for 200 symbols at five words per symbol. To increase this area, input a /MEM directive (section 4.2.5), where each 512-word block will enlarge the capacity of the table by 100 symbols.

INPUTS to the LMGEN comprise:

- Load-module generator directives (section 6.2) input through the SI logical unit.
- Relocatable object modules from which the load module is generated.
- Error-recovery inputs entered via the SI logical unit.

Load-module generator directives define the load module to be generated. They specify the task types (unprotected background or protected foreground) and the locations of the object modules to be used for generation of the load modules. The directives supply information for the cataloging of files, i.e., for storage of the files and the generation of file-directory entries for them. LMGEN directives also provide overlay and loading information. The directives are input through the SI logical unit and listed on the LO logical unit. If the SI logical unit is a Teletype or a CRT device, the message LM** is output on it to indicate that the SI unit is waiting for LMGEN input.

Relocatable object modules are used by LMGEN to generate the load modules. The outputs from both the DAS MR assembler and the FORTRAN compiler are in the form of relocatable object modules. Relocatable object modules can reside on any VORTEX system logical unit and are loaded until an end-of-file mark is found. The last execution address encountered while generating a segment (root or overlay, section 6.1.1) becomes the execution address for that segment. (Note: If the load module being generated is a foreground task, no object module loaded can contain instructions that use addressing modes utilizing the first 2K of memory.

A VORTEX physical record on an RMD is 120 words. Object-module records are blocked two 60-word records per VORTEX physical record. However, in the case of an RMD assigned as the SI logical unit, object modules are not blocked but assumed to be one object module record per physical record.

Error-recovery inputs are entered by the operator on the SI logical unit to recover from errors in load-module generation. Error messages applicable to this component are given in Appendix A.6.

Recovery from the type of error represented by invalid directives or parameters is by either of the following:

a. Input the character C on the SI unit, thus directing LMGEN to go to the SI unit for the next directive.

b. Input the corrected directive on the SI unit for processing. The next LMGEN directive is then input from the SI unit.

If recovery is not desired, input a JCP directive (section 4.2) on the SI unit to abort the LMGEN task and schedule the JCP for execution. (Note: An irrecoverable error, e.g., I/O device failure, causes LMGEN to abort. Examine the I/O error messages and directive inputs to determine the source of such an error.)

OUTPUTS from the LMGEN comprise:

- Load modules generated by the LMGEN
- Error messages
- Load-module maps output upon completion of a load-module generation

Load modules are LMGEN-generated absolute or relocatable tasks with or without overlays. They contain all information required for execution under the VORTEX operating system. During their generation, LMGEN uses the SW logical unit as a work unit. Upon completion of the load-module generation, the module is thus resident on the SW unit. LMGEN can then specify that the module be cataloged on another unit, if required, and output the load module to that unit. Figure 6-1 shows the structure of a load module.
Error messages applicable to the load-module generator are output on the SO and LO logical units. The individual messages, errors, and possible recovery actions are given in Appendix A, section A.6.

Load-module maps are output on the LO logical unit upon completion of the load-module generation, unless suppressed. The maps show all entry and external names and labeled data blocks. They also describe the items given as defined or undefined, and as absolute or relocatable, and indicate the relative location of the items. The load-module map lists the items in the format Four entries per line:

<table>
<thead>
<tr>
<th>Print position</th>
<th>2 3 4 5 6 7 8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>b x b</td>
<td>location</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where

- item is a left-justified entry or external name or labeled data block
- b is a blank
- x is A for an absolute or R for a relocatable item
- location is the left-justified relative location of the item
The following appear at the end of the LMGEN map.

- [SIAP] Top of indirect address pool, which begins at 0500
- [SLIT] Bottom of literal pool, which begins at 0777
- [SLED] Last loaded location. Foreground, word size of load module. Background, last location loaded (loading begins at 01000).

### 6.1.1 Overlays

Load modules can be generated with or without overlays. Load modules with overlays are generated when task requirements exceed core allocation. In this case, the task is divided into overlay segments that can be called as required. Load modules with overlays are generated by use of the OV directive (section 6.2.3) and comprise a root segment and two or more overlay segments (figure 6-1), but only the root segment and one overlay segment can be in memory at any given time. Overlays can contain executable codes, data, or both.

When a load module with overlays is loaded, control transfers to the root segment, which is in main memory. The root segment can then call overlay segments as required.

Called overlay segments may or may not be executed, depending on the nature of the segment. It can be an executable routine, or it can be a table called for searching or manipulation, for example. Whether or not the segment consists of executable data, it must have an entry point.

The generation of the load module begins with the root segment, but overlay segments can be generated in any order.

The root segment can reference only addresses contained within itself. An overlay segment can reference addresses contained within itself or within the root segment. Thus, all entry points referenced within the root segment or an overlay segment are defined for that segment and segments subordinate to it, if any.

For an explanation of DAS MR and FORTRAN calls to overlays see section 2.1.8.

### 6.1.2 Common

Common is the area of memory used by linked programs for data storage, i.e., an area common to more than one program. There are two types of common: named common and blank common. (Refer to the FORTRAN IV Reference Manual, document number 98 A 9902 03x, or the DAS MR COMN directive description in the computer handbook, for the system being used.

Named common is contained within a task and is used for communication among the subprograms within that task.

Blank common can be used like named common or for communication among foreground tasks.

The extent of blank common for foreground tasks is determined at system generation time. The size of the foreground blank common can vary within each task without disturbing the positional relationship of entries but cannot exceed the limits set at system generation time.

The extent of blank common for background tasks is allocated within the load module. The size of the background blank common can vary within each task, but the combined area of the load module and common cannot exceed available memory.

Each blank common is accessible only by the corresponding tasks, i.e., foreground tasks use only foreground blank common, and background tasks use only background blank common.

All definitions of named and blank common areas for a given load module must be in the first object module loaded to generate that load module.

### 6.2 LOAD-MODULE GENERATOR DIRECTIVES

- **TIDB** Create task-identification block
- **LD** Load relocatable object modules
- **OV** Overlay
- **LIB** Library search
- **END**

Load-module generator directives begin in column 1 and comprise sequences of character strings having no embedded blanks. The character strings are separated by commas (,) or by equal signs (=). The directives are free-form and blanks are permitted between the individual character strings of the directives, i.e., before or after commas (or equal signs). Although not required, a period (.) is a line terminator. Comments can be inserted after the period.

The general form of a load-module generator directive is

\[
\text{name}_1, \text{name}_2, \ldots, \text{name}_n
\]

where

- **name** is one of the directive names given above
- **\(p(n)\)** is a parameter required by the directive and defined below under the descriptions of the individual directives
Numerical data can be octal or decimal. Each octal number has a leading zero.

For greater clarity in the descriptions of the directives, optional periods, optional blank separators between character strings, and the optional replacement of commas (,) by equal signs (=) are omitted.

Error messages applicable to load-module generator directives are given in Appendix A.6.

6.2.1 TIDB (Task-Identification Block) Directive

This directive must be input before any other LGMEN directives can be accepted. It permits task scheduling and execution, and specifies the overlay and debugging characteristics of the task. The directive has the general form

TIDB, name, type, segments, DEBUG

where

name is the name (1 to 6 ASCII characters) of the task

type is 1 for an unprotected background task on BL, or 2 for a protected foreground task or 3 for a background task on an alternate library

segments is the number (2 to 9999) of overlay segments in a task with overlays, or 0 for a task without overlays (note that the number 1 is invalid)

DEBUG is present when debugging is desired

The DEBUG parameter includes the DEBUG object module as part of the task. If the task is a load module without overlays, DEBLG is the last object module loaded. If the task is a load module with overlays, DEBUG is the last object module loaded in the root segment (section 6.1.1).

Examples: Specify an unprotected background task named DUMP as having no overlays but with debugging capability.

TIDB, DUMP, 1, 0, DEBUG

Specify a protected foreground task named PROC as having a root segment and four overlay segments.

TIDB, PROC, 2, 4

6.2.2 LD (Load) Directive

This directive specifies the logical unit from which relocatable object modules are to be loaded. It has the general form

LD, lun, key, file

for loading from RMD logical units, and

LD, lun

for loading from any other logical unit, where

lun is the name or number of the logical unit where the object module resides

key is the protection code required to address lun

file is the name of the RMD file

From the object modules, LGMEN generates load modules (with or without overlays) on the SW logical unit. Loading of object modules from the specified logical unit continues until an end-of-file mark is encountered.

Successive LD directives permit the loading of object modules that reside on different logical units.

Examples: Load the relocatable object modules from logical unit 6 (BI) until an end-of-file mark is encountered.

LD, 6

Open a file named DUMP on logical unit 9 (GO) with no protection code. (LGMEN loads the relocatable object modules and closes the file.)

LD, 9,, DUMP

6.2.3 OV (Overlay) Directive

This directive specifies that the named segment is an overlay segment. It has the general form

OV, segname

where segname is the name (1 to 6 ASCII characters) of the overlay segment.

Example: Specify SINE as an overlay segment.

OV, SINE

6.2.4 LIB (Library) Directive

This directive indicates that all load (LD, section 6.2.2) directives have been input, i.e., all object modules have been loaded except those required to satisfy undefined externals. LIB also specifies the libraries to be searched
is an RMD, and thus requires a protection code. The directive has the general form

```
END,lun,key
```

where

- `lun` is the name or number of the logical unit on which the file containing the load module will reside
- `key` is the protection code, if any, required to address `lun`

If TIDB (section 6.2.1) specified an unprotected background task (TIDB directive type = 1), the logical unit, if any, specified by the END directive must be that of the BL unit, i.e., unit 105. If TIDB specified a protected foreground task (TIDB directive type = 2), the logical unit, if any, specified by the END directive must be that of the FL unit, i.e., unit 106, or that of any available assigned RMD partition. If TIDB specified an alternate library background task (TIDB directive type = 3), the logical unit, if any, specified by the END directive, may be that of any available assigned RMD partition.

If the END directive does not specify a logical unit, the load module resides on the SW logical unit only.

If there are still undefined externals, the load module is not catalogued even if END specifies a legal logical unit. In this case, the load module resides on the SW unit only.

```
Examples: Specify that the load module is complete (no more inputs to be made), create a file and a directory entry on the BL logical unit (105), and catalog the module. The protection code is E. (Note: The load module will also reside on the SW unit.)

END, 105, E
```

Specify that the load module is complete (no more inputs to be made) and is to reside on the SW unit only.

```
END
```

### 6.3 Sample Decks for LMGEN Operations

#### Example 1: Card and Teletype Input
Generate a background task without overlays using LMGEN with control records input from the Teletype and object module(s) on cards. Assign the BI logical unit to card reader unit CR00. Assign the task name EXC4 and catalog to the BL logical unit, and load DEBUG as part of the task from the OM library.

```
/JOB, EXAMPLE4
/ASSIGN, BI=CR00
/LMGEN
TIDB, EXC4, 1, 0, DEBUG
LD, BI
LIB
END, BL, E
(ENDJOB)
```
Note: The object module deck must be followed by an end of file (2.78 9 in card column 1).

Example 2: Card Input

Generate a foreground task with overlays using LMGEM with control records and object modules input from the card reader. Assign the BI and SI logical units to card reader unit CR00. Assign the task name EXC5, overlay names SGM1, SGM2, and SGM3, and catalog to the FL logical unit.

```
/JOB,EXAMPLE5
/ASSIGN,BI=CR00,SI=CR00
  (Deck)
  /
/LMGEM
TIDB,EXC5,2,3
LD,BI
  (Object Module(s) -- root segment)
  (End of File)
LIB
OV,SGM1
LD,BI
  (Object Module(s))
  (End of File)
LIB
OV,SGM2
LD,BI
  (Object Module(s))
  (End of File)
LIB
```

Example 3: Teletype and RMD Input

Generate a foreground task without overlays using LMGEM with control records input from the Teletype and object module(s) from an RMD. The object module resides on RMD 107 under the name PGEX. Assign the task name EXC6, search the OM library first to satisfy any undefined externals, and catalog on RMD 120.

```
/JOB,EXAMPLE6
/LMGEM
TIDB,EXC6,2,0
LD,107,Z,PGEX
LIB,OM,D
END,120,X
/ENDJOB
```
SECTION 8
SOURCE EDITOR

The VORTEX operating system source editor (SEDIT) is a background task that constructs sequenced or listed output files by selectively copying sequences of records from one or more input files. SEDIT operates on the principle of forward-merging of subfiles and has file-positioning capability. The output file can be sequenced and/or listed.

8.1 ORGANIZATION

SEDIT is scheduled by the job-control processor (JCP, section 4.2.17) upon input of the JCP directive /SEDIT. Once activated, SEDIT inputs and executes directives from the SI logical unit until another JCP directive (first character = /) is input, at which time SEDIT terminates and the JCP is again scheduled.

SEDIT has a buffer area 'or 100 source records in MOVE operations (section 8.2.8). To increase this, input a /MEM directive (section 4.2.5), immediately preceding the /SEDIT directive, where each 512-word block will increase the capacity of the buffer area by 12 source records.

INPUTS to SEDIT comprise:

a. Source-editor directives (section 8.2) input through the SI logical unit.

b. Old source records input through the IN logical unit.

c. New or replacement source records input through the ALT logical unit.

d. Error-recovery inputs entered via the SO logical unit.

Source-editor directives specify both the changes to be made in the source records, and the logical units to be used in making these changes. The directives are input through the SI logical unit and listed as read on the LO logical unit, with the VORTEX standard heading at the top of each page. If the SI logical unit is a Teletype or a CRT device, the message SE** is output to it before directive input to indicate that the SI unit is waiting for SEDIT input.

There are two groups of source-editor directives: the copying group and the auxiliary group. The copying group directives copy or delete source records input on the IN logical unit, merge them with new or replacement source records input on the ALT unit, and output the results on the OUT unit. Copying-group directives must appear in sequence according to their positioning-record number since there is no reverse positioning. If the remainder of the source records on the IN unit are to be copied after all editing is completed, this must be explicitly stated by an FC directive, (section 8.2.9). Ends of file are output only when specified by FC or WE directives (sections 8.2.9 and 8.2.13). The processing of string-editing directives is different from that of record-editing directives. A string-editing directive affects a specified record where source records on the IN unit are copied onto the OUT unit until the specified record is found and read into memory from the IN unit. After editing, this record remains in memory and is not yet copied onto the OUT unit. This makes possible multiple field-editing operations on a single source record. The auxiliary group directives are those used for special I/O or control functions.

All source records, whether old, new, or replacement records, are arranged in blocks of three 40-word records per VORTEX RMD physical record. Any unused portion of the last physical record of an RMD file on the IN unit should be padded with blanks. When necessary, SEDIT will pad the last RMD record on the OUT unit. When the OUT file will contain more than one source module for input to a language processor, the user should insert two blank records after each END statement to insure that each source module starts on a physical record boundary. Record numbers start with 1 and have a maximum of 9999. Sequence numbers start at any value less than the maximum 9999, and can be increased by any integral increment. These specifications for sequence numbers are given by the SE directive (section 8.2.10).

Error-recovery inputs are entered by the operator on the SO logical unit to recover from errors in SEDIT operations. Error messages applicable to this component are given in Appendix A.8. Recovery is by either of the following:

a. Input the character C on the SO unit, thus directing SEDIT to go to the SI unit for the next directive.

b. Input the corrected directive on the SO unit for processing. The next SEDIT directive is then input from the SI unit.

If recovery is not desired, input a JCP directive (section 4.2) on the SO unit to abort the SEDIT task and schedule the JCP for execution. (Note: If there is an I/O control error on the SO unit, SEDIT is terminated automatically.)

OUTPUTS from the SEDIT comprise:

a. Edited source-record sequences output on the OUT logical unit.

b. Error messages.

c. The listing of the SEDIT directives on the LO logical unit.

d. Comparison outputs (compare-inputs directive, section 8.2.15).

e. Listing of source records on the LO logical unit when specified by the LIST directive (section 3.2.1).
Error messages applicable to SEDIT are output on the SO and LO logical units. The individual messages and errors are given in Appendix A.8.

The listing of the SEDIT directives is made as the directives are read. Source records, when listed, are listed as they are input or output. The VORTEX standard heading appears at the top of each page of the listing.

LOGICAL UNITS referenced by SEDIT are either fixed or reassignable units. The three fixed logical units are:

a. The SI logical unit, which is the normal input unit for SEDIT directives.

b. The SO logical unit, which is used for error processing.

c. The LO logical unit, which is the output unit for SEDIT listings.

The three reassignable logical units are:

a. The SEDIT input (IN) logical unit, which is the normal input unit for source records. This is assigned to the PI logical unit when SEDIT is loaded, but the assignment can be changed by an AS directive with an IN parameter (section 8.2.1).

b. The SEDIT output (OUT) logical unit, which is the normal output unit for source records. This is assigned to the PC logical unit when SEDIT is loaded, but the assignment can be changed by an AS directive with an OU parameter.

c. The SEDIT alternate input (ALT) logical unit, which is the alternate input unit used for new or replacement source records. This is assigned to the BI logical unit when SEDIT is loaded, but the assignment can be changed by an AS directive with an AL parameter.

8.2 SOURCE-EDITOR DIRECTIVES

This section describes the SEDIT directives:

a. Copying group:
   - AS Assign logical units
   - AD Add record($)
   - SA Add string
   - REPL Replace record($)
   - SR Replace string
   - DE Delete record($)
   - SD Delete string
   - MO Move record($)

b. Auxiliary group:
   - FC Copy file
   - SE Sequence records
   - LI List records
   - GA Gang-load all records
   - WE Write end-of-file
   - REWI Rewind
   - CO Compare records

SEdit directives begin in column 1 and comprise sequences of character strings having no embedded blanks. The character strings are separated by commas (,) or by equal signs (=). The directives are free-form and blanks are permitted between individual character strings of the directive, i.e., before or after commas (or equal signs). Although not required, a period (.) is a line terminator. Comments can be inserted after the period.

The general form of an SEDIT directive is

\[ \text{name}, p(1), p(2), \ldots, p(n) \]

where

- name is one of the directive names given above or a longer string beginning with one of the directives names (e.g., AS or ASSIGN)

- each \( p(n) \) is a parameter defined below under the descriptions of the individual directives

Where applicable in the following descriptions, a field specification of the format (first,last) or \( (n1,n2,n3) \) is still separated from other parameters by parentheses even though it is enclosed in commas. Note also that the character string string is coded within single quotation marks, which are, of course, neither a part of the string itself nor of the character count for the string.

8.2.1 AS (Assign Logical Units) Directive

This directive specifies a unit assignment for an SEDIT reassignable logical unit (section 8.1). It has the general form

\[ \text{AS}, \text{nn} = \text{lun}, \text{key}, \text{file} \]

where

- \( \text{nn} \) is IN if the directive is making an assignment of the IN logical unit, OU if the OUT logical unit, or AL if the ALT logical unit

- lun is the name or number of the logical unit being assigned as the IN, OUT, or ALT unit

- key is the protection code, if any, required to address lun

- file is the name of an RMD file, if required

If the SEDIT reassignable units are to retain the assignments made when SEDIT was loaded (default assignments: IN = PI, OUT = PO, ALT = BI), no AS direc-
8.2.3 SA (Add String) Directive

This directive inserts a character string into a source-record field. It has the general form

\[ \text{SA, recno, (first, last), 'string'} \]

where

- \( \text{recno} \) is the number of the source record in which the character string is to be inserted
- \( \text{first} \) is the number of the first character position to be affected
- \( \text{last} \) is the number of the last character position to be affected
- \( \text{string} \) is the string of characters to be inserted in the field delimited by character positions \( \text{first} \) and \( \text{last} \) or record number \( \text{recno} \)

The SA directive copies source records from the \( \text{IN} \) logical unit onto the \( \text{OUT} \) logical unit beginning with the current position of the \( \text{IN} \) unit and continuing up to but not including the record specified by \( \text{recno} \). The record \( \text{recno} \) is read into the memory buffer. The character string string shifts into the left end of the specified field \( \text{first, last} \), with characters shifted out of the right end of the field being lost. There is no check on the length of string and shifting continues until it is left-justified in the field with excess characters, if any, being truncated on the right.

The record remains in the memory buffer, thus permitting multiple string operations on the same record. (If \( \text{IN} \) is already positioned at \( \text{recno} \) because of a previous string operation, there is, of course, no change in position.)

The record \( \text{recno} \) is read out of the memory buffer and onto the \( \text{OUT} \) unit when an SEDIT directive affecting another record is input.

The field specification \( \text{first, last} \) is lost after one manipulation. Subsequent string operations must specify the character positions based on the new configuration. For example, for the character string ACDEGbb in positions 1 through 7, addition of the character B in position 2 requires the field specification (2, 7). Then, to add the character F between E and G, one must specify the field (5, 7) rather than (5, 7) because of the shift previously caused by insertion of the character B.

Example: Change the erroneous DAS MR source statement operand in character positions 16-21 of the 32nd record from LOCXbb to LOCXb.

\[ \text{SA, 32, (19, 20), ', '} \]
8.2.4 REPL (Replace Records) Directive

This directive replaces one sequence of source records with another sequence of records. It has the general form

REPL, recno1, recno2

where

recno1 is the number of the first record to be replaced

recno2 is the number of the last record to be replaced

If recno2 is omitted, it is assumed equal to recno1, i.e., one record will be replaced.

The REPL directive copies source records from the IN logical unit onto the OUT logical unit beginning with the current position of the IN unit and continuing up to but not including the record specified by recno1. Then, records are read from IN, but not copied onto OUT, up to and including the record specified by recno2. Thus, the records recno1 through recno2, inclusive, are deleted. Then, source records are copied from the ALT logical unit from the current position of the unit up to but not including the next end-of-file mark.

Example: Copy records from IN onto OUT from the current position of IN up to and including record 9. Replace IN records 10 through 20, inclusive, with records on ALT, copying those between the current position of ALT and the next end-of-file mark onto OUT. Do not copy the end-of-file mark.

REPL, 10, 20

8.2.5 SR (Replace String) Directive

This directive replaces one character string within a source record with another character string. It has the general form

SR, recro, (n1, n2, n3), 'string'

where

recro is the number of the source record in which the character string is to be replaced

n1 is the number of the first character position of the string to be replaced

n2 is the number of the last character position of the string to be replaced

n3 is the number of the last character position of the field in which the string to be replaced occurs

string is the string of characters to be inserted in the field delimited by character positions n1 and n3 in record number recro after shifting out the characters in positions n1 through n2, inclusive.

The SR directive copies source records from the IN logical unit onto the OUT logical unit beginning with the current position of the IN unit and continuing up to but not including the record specified by recro. The record recro is read into the memory buffer. Field n1, n3 is then shifted to the left and filled with blanks until the field n1, n2 is shifted out. Then, the character string string shifts into the left end of the field n1, n3. There is no check on the length of string and shifting continues until it is left-justified in the field n1, n3 with excess characters, if any, being truncated on the right.

The record remains in the memory buffer, thus permitting multiple string operations on the same record. (If IN is already positioned at recro because of a previous string operation, there is, of course, no change in position.)

The record recro is read out of the memory buffer and onto the OUT unit when a SEDIT directive affecting another record is input.

The field specification n1, n2, n3 is lost after one manipulation. Subsequent string operations must specify the character positions based on the new configuration.

Example: Copy records from IN onto OUT up to and including record 49, and replace the present contents of character positions 10 through 12, inclusive, in IN unit source record 50 with the character string XYb.

SR, 50, (10, 12, 12), 'XYb'

8.2.6 DE (Delete Records) Directive

This directive deletes a sequence of source records. It has the general form

DE, recno1, recno2

where

recno1 is the number of the first record to be deleted

recno2 is the number of the last record to be deleted

If recno2 is omitted, it is assumed equal to recno1, i.e., one record will be deleted.
The DE directive processing is exactly like that of the REPL directive (section 8.2.4) except that there is no copying from the ALT unit after the deletion of the records recno1 through recno2, inclusive.

**Examples:** Copy records from IN onto the OUT logical unit up to and including record 49, but delete records 50 through 54, inclusive.

```
DE, 50, 54
Position IN at record 2, deleting record 1.
DE, 1
```

### 8.2.7 SD (Delete String) Directive

This directive deletes a character string from a source record. It has the general form

```
SD,recno,(n1,n2,n3)
```

where

- **recno** is the number of the source record from which the character string is to be deleted.
- **n1** is the number of the first character position of the string to be deleted.
- **n2** is the number of the last character position of the string to be deleted.
- **n3** is the number of the last character position of the field in which the string to be deleted occurs.

The SD directive processing is exactly like that of the SR directive (section 8.2.5) except that now new character string is shifted into field n2,n3 after the field n1,n2 is shifted out.

**Example:** Copy records from IN onto OUT up to and including record 99, and delete characters 2 through 4, inclusive, from record 100, shifting characters 5 through 10, inclusive, three places to the left, with blank fill on the right.

```
SD, 100,(2,4,10)
```

### 8.2.8 MO (Move Records) Directive

This directive moves a block of records forward on a unit. It has the general form

```
MO,recno1,recno2,recno3
```

where

- **recno1** is the number of the first record to be moved.
- **recno2** is the number of the last record to be moved.
- **recno3** is the number of the record after which the block of records delimited by recno1 and recno2 is to be inserted.

If recno2 is omitted, it is assumed equal to recno1, i.e., one record will be moved.

The MO directive copies source records from the IN logical unit onto the OUT logical unit beginning with the current position of the IN unit and continuing up to but not including the record specified by recno1. The records recno1 through recno2 are then read into a special MOVE area in memory. The position of IN is row recno2 + 1. When OUT reaches (by some succeeding directive) recno3 + 1, the contents of the MOVE area are copied onto OUT. Multiple MO operations are legal.

**Example:** Copy records from IN onto OJT up to and including record 4, save records 5 through 10, inclusive, in the MOVE area of memory, copy records 11 through 99, inclusive, from IN onto OUT, and then copy records 5 through 10 from the MOVE area to OUT. This gives a record sequence on OUT of 1.4, 11.99, 5-10 (FC directive, section 8.2.9.).

```
MO, 5, 10, 99
FC
```

### 8.2.9 FC (Copy File) Directive

This directive copies blocks of files, including end-of-file marks. It has the general form

```
FC,nfiles
```

where **nfiles** (default value = 1) is the number of files to be copied.

If the IN logical unit and/or the OUT logical unit is an RMD partition, nfiles must be 1 or absent. If OUT is a named file on an RMD, there will be an automatic close/update. Whenever an end-of-file mark is encountered, all record counters are reset to zero.

```
FC, 6
```
Examples: Copy files from IN onto OUT up to and including the next end-of-file mark on the IN unit.

FC
Copy the next six IN files (including end-of-file marks) onto OUT. This includes the sixth end-of-file mark. (Note: If IN and/or OUT is an RMD partition, there will be an error.)

FC, 6

8.2.10 SE (Sequence Records) Directive

This directive assigns a decimal sequence number to each source record output to the OUT logical unit. It has the general form

SE,(first,last),initial,increment

where
first is the first character position of the sequence name field
last is the last character position of the sequence number field, where the default value of first,last is 76,80
initial is the initial number to be used as a sequence number (default value = 10)
increment is the increment to be used between successive sequence numbers (default value = 10)

There is also a special form of the SE directive to stop sequencing:

SE,N

where there are no parameters other than the letter N.

Examples: In the next record output to OUT, place 00010 in character positions 76 through 80, and increment the field by 10 in each succeeding record.

SE
In the next record output to OUT, place 030 in character positions 15 through 17, and increment the field by 7 on each succeeding record.

SE,(15,17),30,7
Stop sequencing.

SE,N

8.2.11 LI (List Records) Directive

This directive lists, on the LO logical unit, the records copied onto the OUT unit. The L directive has the general form

LI,list

where list is A (default value) if all OUT records are to be listed, C if only changed records are to be listed, or N if listing is to be suppressed. Source records output to the OUT file are listed with their OUT record number at the left of the print list.

Examples: List all records output to OUT.

LI

Suppress all listing except that of SEDIT directives.

LI,N

8.2.12 GA (Gang-Load All Records) Directive

This directive loads the same character string into the specified field of every record copied onto the OUT logical unit. It has the general form

GA,(first,last),'string'

where
first is the first character position of the field to be gang-loaded
last is the last character position of the field to be gang-loaded, where the default value of first,last is 73,75
string is the string of characters to be gang-loaded into character positions first through last, inclusive in all records copied onto out

There is also a special form of the GA directive to stop gang-loading:

GA

where there are no parameters in the directive.

In every OUT record, GA clears the specified field, and loads the string into it. There is no check on the length of string and shifting continues until it is left-justified in the specified field with excess characters, if any, being truncated on the right.
Examples: Load character string VDMbb in character positions 11 through 15, inclusive, of every record copied onto OUT.

GA, (11, 15), 'VDM'

Stop gang-loading.

GA

8.2.13 WE (Write End of File) Directive

This directive writes an end-of-file mark on the OUT logical unit. It has the form

WE

without parameters. If OUT is a named file on an RMD, there will be an automatic close/update.

Example: Write an end-of-file mark on OUT, a magnetic-tape unit.

WE

8.2.14 REWI (Rewind) Directive

This directive rewinds the specified SEDIT logical unit(s). It has the general form

REWIp(1),p(2),p(3)

where each p(n) is a name of one of the SEDIT logical units: IN, OUT, or ALT. These can be coded in any order.

Example: Rewind all SEDIT logical units.

REWl, IN, ALT, OUT

8.2.15 CO (Compare inputs) Directive

This directive compares the specified field in the inputs from the IN logical unit with those from the ALT logical unit and lists discrepancies on the LO logical unit. The directive has the general form

CO,(first,last),limit

where

first is the first character position of the field to be compared

last is the last character position of the field to be compared, where the default value of first,last is 1,80.

limit is the maximum number of discrepancies to be listed before aborting the comparison and passing to the next directive.

Any discrepancy between the IN and ALT inputs is listed in the format

I recordnumber or EOF inrecord
A recordnumber or EOF altrecord

If the comparison terminates by reaching the limit number of discrepancies, SEDIT outputs on the LO the message

SEDITION ABORTED

to prevent long listings of errors, for example, where a card is misplaced or missing on one input. A normal termination of a comparison (at the next end-of-file mark) concludes with the message

SEDITION COMPLETED

Example: Compare character positions 1 through 80, inclusive, from the IN and ALT units until either an end of file is found or there have been 5 discrepancies listed on the LO.

CO,5

8.3 EXAMPLE OF EDITING A FILE

Following is a sample job stream for editing an existing file on a magnetic tape onto a new file on magnetic tape. The input file consists of 80-character records followed by an end-of-file mark. The job stream and the edit cards are read through the system input device.

/JOB,EDIT
/ASSIGN,PI=MT00,PO=MT10
/REW,PI,PO
/SEDITION
AS, IN=PI
AS,OUT=PO
AS,ALT=SI
DE,5
REPL,8,10
LDA TEMP
(EIF card, 2-7-8-9 punch)
ADD,17
TBL  BSS 5
(EIF card, 2-7-8-9 punch)
FC
REWl,IN,OUT
/ENDJOB

8.7
The result of running the preceding source editor example would be the following:

<table>
<thead>
<tr>
<th>Input File</th>
<th>Output File</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 *</td>
<td>1 *</td>
</tr>
<tr>
<td>2 *</td>
<td>2 *</td>
</tr>
<tr>
<td>3 *</td>
<td>3 *</td>
</tr>
<tr>
<td>4 A$3 EQU 6</td>
<td>4 A$3 EQU 6</td>
</tr>
<tr>
<td>5 B$3 EQU 9</td>
<td>5 *</td>
</tr>
<tr>
<td>6 *</td>
<td>6 CATLOG DATA 0</td>
</tr>
<tr>
<td>7 CATLOG DATA 0</td>
<td>7 LDA TEM?</td>
</tr>
<tr>
<td>8 LDA THX</td>
<td>8 ADD PARM6</td>
</tr>
<tr>
<td>9 LDB THY</td>
<td>9 ANAI 0770</td>
</tr>
<tr>
<td>10 JBZM ODER</td>
<td>10 STA TBL+2</td>
</tr>
<tr>
<td>11 ADD PARM6</td>
<td>11 LRLA 6</td>
</tr>
<tr>
<td>12 ANAI 0770</td>
<td>12 STA TBL+4</td>
</tr>
<tr>
<td>13 STA TBL+2</td>
<td>13 T2B</td>
</tr>
<tr>
<td>14 LRLA 6</td>
<td>14 JMP* CATLOG</td>
</tr>
<tr>
<td>15 STA TBL+4</td>
<td>15 TBL BSS 5</td>
</tr>
</tbody>
</table>
SECTION 9
FILE MAINTENANCE

The VORTEX file-maintenance component (FMAIN) is a background task that manages file-name directories and the space allocations of the files. It is scheduled by the job-control processor (JCP) upon input of the JCP directive /FMAIN (section 4.2.18).

Only files assigned to rotating-memory devices (disc or drum) can be referenced by name.

File space is allocated within a partition forward in contiguous sectors of the same cylinder, skipping bad tracks. The only exception to this continuity is the file-name directory itself, which is a sequence of linked sectors that may or may not be contiguous.

9.1 ORGANIZATION

FMAIN inputs file-maintenance directives (section 9.2) received on the SI logical unit and outputs them on the LO logical unit and on the SO logical unit if it is a different physical device from the LO unit. Each directive is completely processed before the next is input to the JCP buffer.

If the SI logical unit is a Teletype or a CRT device, the message FM** is output on it before input to indicate that the SI unit is waiting for FMAIN input.

If there is an error, one of the error messages given in Appendix A.9 is output on the SO logical unit, and a record is input from the SO unit to the JCP buffer. If the first character of this record is /, FMAIN exits via the EXIT macro. If the first character is C, FMAIN continues. If the first character is neither / nor C, the record is processed as a normal FMAIN directive. FMAIN continues to input and process records until one whose first character is / is detected, when FMAIN exits via exit. (An entry beginning with a carriage return is an exception to this, being processed as an FMAIN directive).

FMAIN has a symbol-table area for 200 symbols at five words per symbol. To increase this area, input a /MEM directive (section 4.2.5), where each 512-word block will enlarge the capacity of the table by 100 symbols.

9.1.1 Partition Specification Table

Each rotating-memory device (RMD) is divided into up to 20 memory areas called partitions. Each partition is referenced by a specific logical-unit number. The boundaries of each partition are recorded in the core-resident partition specification table (PST). The first word of the PST contains the number of VORTEX physical records per track. The second word of the PST contains the address of the bad-track table, if any. Subsequent words in the PST comprise the four-word partition entries. Each PST is in the format:

<table>
<thead>
<tr>
<th>Bit</th>
<th>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 0</td>
<td>Number of 120-word logical records per track</td>
</tr>
<tr>
<td>Word 1</td>
<td>Address of bad tracks table (0 if none)</td>
</tr>
<tr>
<td>Word 0</td>
<td>Beginning partition track address</td>
</tr>
<tr>
<td>Word 1</td>
<td>PPB</td>
</tr>
<tr>
<td>Word 2</td>
<td>Number of bad tracks in partition</td>
</tr>
<tr>
<td>Word 3</td>
<td>Ending partition address + 1</td>
</tr>
</tbody>
</table>

The partition protection bit, designated ppb in the above PST entry map, is unused in file maintenance procedures.

Note that PST entries overlap. Thus, word 0 of each PST entry is also word 0 of the following entry. The relative position of each PST entry is recorded in the device specification table (DST) for that partition.

The bad-track table, whose address is in the second word of the PST, is a bit string read from left to right within each word, and forward through contiguous words, with set bits flagging bad tracks on the RMD. (If there is no bad-track table, the second word of the PST contains zero.)

9.1.2 File-Name Directory

Each RMD partition contains a file-name directory of the files contained in that partition. The beginning of the directory is in the first sector of the partition. The directory for each partition has a variable number of entries arranged in n sectors, 19 entries per sector. Sectors containing directory information are chained by pointers in
the last word of each sector. Thus, directory sectors need not be contiguous. Each directory entry is in the format:

<table>
<thead>
<tr>
<th>Bit</th>
<th>5 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 0</td>
<td>File name</td>
</tr>
<tr>
<td>Word 1</td>
<td>File name</td>
</tr>
<tr>
<td>Word 2</td>
<td>File name</td>
</tr>
<tr>
<td>Word 3</td>
<td>Current position of file</td>
</tr>
<tr>
<td>Word 4</td>
<td>Beginning file address</td>
</tr>
<tr>
<td>Word 5</td>
<td>Ending file address</td>
</tr>
</tbody>
</table>

The file name comprises six ASCII characters packed two characters per word, left justified, with blank fill. Word 3, which contains the current address at which the file is positioned, is initially set to the ending file address, and is manipulated by I/O control macros (section 3). The extent of the file is defined by the addresses set in words 4 and 5 when the file is created, and remains constant.

The first sector of each partition is assigned to the file-name directory. FMAIN allocates RMD space forward in contiguous sectors, skipping bad tracks. Following the last entry in each sector is a one-word tag containing either the value 01 (end of directory), or the address of the next sector of the file-name directory.

The file-name directories are created and maintained by the file-maintenance component for the use of the I/O control component (section 3). User access to the directories is via the I/O control component.

Special entries: A blank entry is created when a file name is deleted, in which case the file name is ***** and words 3 through 5 give the extent of the blank file. A zero entry is created when one name of a multivariate file is deleted, in which case the deleted name is converted to a blank entry and all other names of the multivariate file are set to zero.

**WARNING**

To prevent possible loss of data from the file-name directory during file-maintenance operations, FMAIN sets the lock bit (bit 12 of word 2 of the DST, section 3.2) before any directory operation, thus inhibiting all foreground requests for I/O with the partition being modified. Upon completion of the directory operation, FMAIN clears the lock bit. Except for the use of protection codes, this is the only protection for the file-name directory. Manipulation of foreground files with FMAIN is at the user's risk. For example, VORTEX does not prevent deletion of a file name from a file-name directory that has been opened and is being written into by a foreground program. Therefore, foreground files should be reassigned prior to manipulation by FMAIN.

### 9.1.3 Relocatable Object Modules

Outputs from both the DAS MR assembler and the FORTRAN compiler are in the form of relocatable object modules. Relocatable object modules can reside on any VORTEX-system logical unit. Before object modules can be read from a unit by the FMAIN INPUT and ADD directives (sections 9.2.7 and 9.2.8), an I/O OPEN with rewinding (section 3.5.1) is performed on the logical unit, i.e., the unit (except paper tape or card readers) is first positioned to the beginning of device or load point for that unit. Object modules can then be loaded until an end-of-file mark is found.

The system generator (section 15) does not build any object-module library. FMAIN is the only VORTEX component used for constructing user object-module libraries.

A VORTEX physical record on an RMD is 120 words. Object-module records are blocked two 60-word records per VORTEX physical record. However, in the case of an RMD assigned as the SI logical unit, object modules are not blocked but assumed to be one object-module record per physical record.

### 9.1.4 Output Listings

FMAIN outputs four types of listing to the LO logical unit:

- **Directive listing** lists, without modification, all FMAIN directives entered from the SI logical unit.
- **Directory listing** lists file names from a logical unit file-name directory in response to the FMAIN directive LIST (section 9.2.5).
- **Deletion listing** lists file names deleted from a logical unit file-name directory in response to the FMAIN directive DELETE (section 9.2.2).
- **Object-module listing** lists the object-module input in response to the FMAIN directive ADD (section 9.2.8).

All FMAIN listings begin with the standard VORTEX heading.

The directory listing is further described under the discussion of FMAIN directive LIST (section 9.2.5), the deletion listing under DELETE (section 9.2.2), and the object-module listing under ADD (section 9.2.8).

### 9.2 FILE-MAINTENANCE DIRECTIVES

This section describes the file-maintenance directives:

- **CREATE file**  
- **DELETE file**  
- **RENAME file**  
- **ENTER new file name**  
- **LIST file names**  
- **INIT (initialize) directory**
  - **INPUT logical unit for object module**
  - **ADD object module**
File maintenance directives comprise sequences of character strings having no embedded blanks. The character strings are separated by commas (,) or by equal signs (=). The directives are free form and blanks are permitted between the individual character strings of the directive, i.e., before or after commas (or equal signs). Although not required, a period (.) is a line terminator. Comments can be inserted after the period.

The general form of a file maintenance directive is

```
directive, lun, p(1), p(2), ... , p(n)
```

where

- `directive` is one of the directives listed above in capital letters
- `lun` is the number or name of the affected logical unit
- `p(n)` is a parameter defined under the descriptions of the individual directives below

Numerical data can be octal or decimal. Each octal number has a leading zero.

For greater clarity in the descriptions of the directives, optional periods, optional blank separators between character strings, and the optional replacement of commas (,) by equal signs (=) are omitted.

Error messages applicable to file-maintenance directives are given in Appendix A.9.

### 9.2.2 DELETE Directive

This directive deletes the designated file and all file-name directory references to it from the specified logical unit. It converts the specified file-name directory entry to a blank entry (name field = "******", section 9.1.2) and all other directory references to this file to zero entries (all fields = zero, section 9.1.2), and outputs a listing of deleted file names on the LO logical unit. The directive has the general form

```
DELETE, lun, key, name
```

where

- `lun` is the number or name of the logical unit from which the file is being deleted
- `key` is the protection code, if any, required to address `lun`
- `name` is the name of the file being deleted (in the case of a multiname file, any one of the names can be used)

The output format has, following the FMAIN heading, a two-line heading

```
DELETE LISTING FOR lun
FILE NAME START END CURRENT
```

where `lun` is the number of the logical unit from which the file is being deleted. This heading is followed by a blank line and a listing of all file-names being deleted, one per line. Words 0-2 of the file-name directory entry (section 9.1.2) are placed in the FILE NAME column; word 3, (in octal) in the CURRENT column; word 4, (in octal) in the START column; and word 5, (in octal) in the END column. After the last file name, there is an entry describing the blank file created by the deletion, where the FILE NAME column contains "******", the START column contains the next available address (word 2 of the PST entry), and both the CURRENT and END columns contain the last address + 1 (word 3 of the PST entry).
Example: Delete the file ZFILE (and all file-name directory entries referencing it) from logical unit 112, whose protection code is P.

DELETE, 112, P, ZFILE

The name ZFILE is replaced in the file-name directory by *****, and the space allocation for this blank entry extended in both directions to include adjacent blank entries, if any. Any blank entries thus absorbed are converted to zero entries, as are all other entries that reference the file ZFILE. All affected file-name directory entries are listed on the LO logical unit.

9.2.3 RENAME Directive

This directive changes the name of a file, but does not otherwise modify the file-name directory. The directive has the general form

RENAME, lun, key, old, new

where
lun is the number or name of the logical unit where the file to be renamed is located
key is the protection code, if any, required to address lun
old is the old name of the file being renamed
new is the new name of the file being renamed

Following RENAME, old can no longer be used to reference the file.

Example: On logical unit 112, whose protection code is P, change the name of the file XFILE to YFILE.

RENAME, 112, P, XFIRE, YFILE

9.2.4 ENTER Directive

This directive adds a new file name to be used in referencing an existing file, but does not otherwise modify the file-name directory. ENTER thus permits multiname access to a file. The directive has the general form

ENTER, lun, key, old, new

where
lun is the number or name of the logical unit where the affected file is located
key is the protection code, if any, required to address lun
old is an old name of the affected file
new is the new name by which the file can also be referenced

Example: On logical unit 113, whose protection code is K, make the file X1 accessible by using either the name X1 or the name Y1.

ENTER, 113, K, X1, Y1

9.2.5 LIST Directive

This directive outputs on the LO logical unit the file-name directory of the specified logical unit. The output comprises the file names, file extents, current end-of-file positions, logical-unit name or number, and the extent of unassigned space in the partition. All number are in octal. The directive has the general form

LIST, lun, key

where
lun is the number or name of the logical unit whose contents are to be listed
key is the protection code, if any, required to address lun

The output format has a two-line heading

FILE DIRECTORY FOR LUN lun
FILE NAME START END CURRENT

where lun is the number or name of the logical unit whose contents are being listed. This heading is followed by a blank line and a listing of all file names from the directory, one name per line. Words 02 of the file-name directory entry (section 9.1.2) are placed in the FILE NAME column; word 4, (in octal) in the START column; word 3, (in octal) in the END column; and word 5, (in octal) in the CURRENT column. After the last file name, if there is any unassigned space in the partition, there is an entry describing the unassigned space in the partition, where the FILE NAME column contains "UNAS", the START column contains the next available address (word 2 of the PST entry), and both the CURRENT and END columns contains the last address + 1 (word 3 of the PST entry).

Example: List the file-name directory of logical unit 114, which has no protection code.

LIST, 114

9.2.6 INIT (Initialize) Directive

This directive clears the entire file-name directory of the specified logical unit, deletes all file names in it, and releases all currently allocated file space in the partition by reducing the file-name directory to a single end-of-directory entry. The directive has the general form

INIT, lun, key

where
lun is the number or name of the logical unit being initialized
key is the protection code, if any, required to address lun
Example: Initialize the file-name directory on logical unit 115, which has protection code X.

\[ \text{INIT, 115, X} \]

9.2.7 INPUT Directive

This directive specifies the logical unit from which object modules are to be input. Once specified, the input logical unit number is constant until changed by a subsequent INPUT directive. The directive has the general form

\[ \text{INPUT, lun, key, file} \]

where

- \( \text{lun} \) is the number or name of the logical unit from which object modules are to be input
- \( \text{key} \) is the protection code, if any, required to address lun
- \( \text{file} \) is the name of the RMD file containing the required object module(s)

Neither key nor file are required unless lun is a RMD partition.

NOTE

There is no default value. Thus, if an attempt is made to input an object module (ADD directive, section 9.2.8) without defining the input logical unit by an INPUT directive, an error message will be output.

Examples: Specify logical unit 6 as the device from which object modules are to be input.

\[ \text{INPUT, 6} \]

Open and rewind the file ARCTAN on logical unit 104, which has protection code D.

\[ \text{INPUT, 104, D, ARCTAN} \]

9.2.8 ADD Directive

This directive reads object modules from the INPUT unit (section 9.2.7) and writes them onto the SW logical unit, checking for entry names and validating check sums, record sizes, loader codes, sequence numbers, and record structures. Reading continues until an end of file is encountered. Entry names are then added to the file-name directory of the specified logical unit and the object modules are copied from the SW logical unit onto the specified logical unit. The directive has the general form

\[ \text{ADD, lun, key} \]

where

- \( \text{lun} \) is the number or name of the logical unit onto which object modules are to be written
- \( \text{key} \) is the protection code, if any, required to address lun

The specified logical unit lun references a system or user object-module library.

The names of the object modules and their date of generation, size in words (zero for FORTRAN modules), entry names, and referenced external names are listed on the LO logical unit.

To recover from errors in object-module processing, reposition the logical unit to the beginning of the module.

Example: Add object modules to logical unit 104, which has protection code D.

\[ \text{ADD, 104, D} \]
SECTION 17
OPERATOR COMMUNICATION

The operator communicates with the VORTEX system through the operator communication component by means of operator key-in requests input through the operator communication (OC) logical unit.

17.1 DEFINITIONS

An operator key-in request is a string of up to 80 characters beginning with a semicolon. The request is initiated by the operator and is input through the OC unit. An operator key-in request is independent of I/O requests via the IOC (section 3) and, hence, is known as an unsolicited request.

The operator communication (OC) logical unit is the logical unit through which the operator inputs key-in requests. There is only one OC unit in the VORTEX system. Initially, the OC unit is the first Teletype, but this assignment can be changed by use of the ;ASSIGN key-in request (section 17.2.9).

17.2 OPERATOR KEY-IN REQUESTS

This section describes the operator key-in requests:

- ;SCHED Schedule foreground task
- ;TSCHED Time-schedule foreground task
- ;ATTACH Attach foreground task to PIM line
- ;RESUME Resume task
- ;TIME Enter or display time-of-day
- ;DATE Enter date
- ;ABORT Abort task
- ;TSTAT Test task status
- ;ASSIGN Assign logical unit(s)
- ;DEVON Device down
- ;DEVUP Device up
- ;IOLIST List logical-unit assignments

where

<table>
<thead>
<tr>
<th>request</th>
<th>is one of the key-in requests listed above in capital letters</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(n)</td>
<td>is a parameter defined under the descriptions of the individual key-in requests below</td>
</tr>
<tr>
<td>cr</td>
<td>is the carriage return, which terminates all operator key-in requests</td>
</tr>
</tbody>
</table>

Each operator key-in request begins with a semicolon (;) and ends with a carriage return. Parameters are separated by commas. A backarrow (\) deletes the preceding character. A backslash (\) deletes the entire present key-in request.

Table 17-1 shows the system names of physical I/O devices as used in operator key-in requests.

For greater clarity, optional blank separators between character strings, and the optional replacement of commas (,) by equal signs (=) are omitted from the descriptions of the key-in requests.

Error messages applicable to operator key-in requests are given in Appendix A.17.

<table>
<thead>
<tr>
<th>Table 17-1. Physical I/O Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Name</td>
</tr>
<tr>
<td>------------</td>
</tr>
<tr>
<td>DUM</td>
</tr>
<tr>
<td>CPCu</td>
</tr>
<tr>
<td>CRcu</td>
</tr>
<tr>
<td>CTcu</td>
</tr>
<tr>
<td>Dcup</td>
</tr>
<tr>
<td>LPCu</td>
</tr>
<tr>
<td>MTcu</td>
</tr>
<tr>
<td>PTCu</td>
</tr>
<tr>
<td>TYcu</td>
</tr>
<tr>
<td>Clima, COma</td>
</tr>
</tbody>
</table>
NOTES

c = Controller number. For each type of device, controllers are numbered from 0 as required.

u = Unit number. For each controller, units are numbered from 0 as required (within the capacity of the controller).

cu can be omitted to specify unit 0 controller 0, e.g., CR00 or CR.

p = Partition letter. RMD partitions are lettered from A to T as required to refer to a partition on the specified device, e.g., D00A.

17.2.1 ;SCHED (Schedule Foreground Task) Key-In Request

This key-in request immediately schedules the specified foreground-library task for execution at the designated priority level. It has the general form

;SCHED,task,level,lun,key

where

- task is the name of the foreground task to be scheduled
- level is the priority level (from 2 to 31) of the scheduled task
- lun is the number or name of the foreground-library rotating-memory logical unit where the scheduled task resides (0 for scheduling a resident foreground task)
- key is the protection code, if any, required to address lun
- time is the scheduled time in hours (from 00 to 23) and minutes (from 00 to 59), e.g., 1945 for 7:45 p.m.

Operator key-in examples. Schedule on priority level 3 the foreground task DOTASK residing on the US logical unit. Use T as the protection key.

;SCHED,DOTASK,3,US,T,2330

Schedule for execution at 8:30 a.m. on priority level 9 the resident foreground task TESTIO.

;SCHED,TESTIO,9,0,0,0830

17.2.2 ;TSCHE D (Time-Schedule Foreground Task) Key-In Request

This key-in request schedules the specified foreground-library task for execution at the designated time-of-day and priority level. It has the general form

;TSCHE D,task,level,lun,key,time

where

- task is the name of the foreground task to be scheduled
- level is the priority level (from 2 to 31) of the scheduled task
- lun is the number or name of the foreground-library rotating-memory logical unit where the scheduled task resides (0 for scheduling a resident foreground task)
- key is the protection code, if any, required to address lun
- time is the scheduled time in hours (from 00 to 23) and minutes (from 00 to 59), e.g., 1945 for 7:45 p.m.

Operator key-in examples. Schedule on priority level 3 the foreground task DOTASK residing on the US logical unit. Use T as the protection key.

;TSCHED,DOTASK,3,US,T,2330

Schedule for execution at 8:30 a.m. on priority level 9 the resident foreground task TESTIO.

;TSCHED,TESTIO,9,0,0,0830

17.2.3 ;ATTACH Key-In Request

This key-in request attaches the specified foreground task to the designated PIM (priority interrupt module) line. It has the general form

;ATTACH,task,line,iow.enable

where

- task is the name of the foreground task to be attached to the PIM line
- line is the two-digit number of the PIM line to which the task is to be attached, with the
tens digit specifying the PIM number (1-8) and the units digit the line number (0-7) on that PIM

lew is the value (from 01 to 0177777) of the interrupt event word (section 17) and identifies the bit(s) to be set in the task TIDB when an interrupt occurs on line

enable is E (default value) to enable the line, or D to disable it

The task can be resident or nonresident. However, its TIDB must have been defined at system-generation time. ATTACH provides a flexible way of altering interrupt assignments without having to regenerate the system.

Operator key-in example: Connect task INTRPT to PIM 1, line 3. Use 020 as the interrupt event word value (i.e., set bit 4 of the interrupt event word in TIDB if INTRPT is scheduled due to an interrupt on PIM 1, line 3).

;ATTACH, INTRPT, 13, 020

A PIM directive with the PIM line to be attached must have been specified during system generation to set up the link to the interrupt line handler region.

17.2.4 ;RESUME Key-In Request

This key-in request reactivates the specified task for execution at its specified priority level. It has the general form

;RESUME, task

where task is the name of the task to be resumed

Operator key-in example: Resume the task DOTASK.

;RESUME, DOTASK

17.2.5 ;TIME Key-In Request

This key-in request enters the specified time, if any, as system time-of-day. If no time is specified in the key-in request, ;TIME displays the current time-of-day. The key-in request has the general form

;TIME time

where time is the time-of-day in hours (from 00 to 23) and minutes (from 00 to 59), e.g., 1945 for 7:45 p.m.

The time-of-day output for a ;TIME request without time is of the form

T hhmm HRS

where hhmm is the time of day in hours and minutes.

Operator key-in example: Set the system time-of-day to 3:00 p.m.

;TIME, 1500

17.2.6 ;DATE Key-In Request

This key-in request enters the specified date as the system date. It has the general form

;DATE, mm/dd/yy

where

mm is the month (00 to 12)

dd is the day (00 to 31)

yy is the year (00 to 99)

Note that since the entire date is considered one parameter, there are no commas other than the one immediately following DATE. The components of the date are, however, separated by slashes as shown.

Operator key-in example: Set the system date to 25 December 1971.

;DATE, 12/25/71

17.2.7 ;ABORT Key-In Request

This key-in request aborts the specified task. It has the general form

;ABORT, task

where task is the name of the task to be aborted

Operator key-in example: Abort the task DOTASK.

;ABORT, DOTASK

17.2.8 ;TSTAT (Task Status) Key-In Request

This key-in request outputs the status of the specified task, if any. If no task is specified, ;TSTAT outputs the status of all tasks queued on the active task identification block
(TIDB) stack. This request is not applicable to tasks having no established TIDB. The request has the general form

\[ \text{\texttt{;TSTAT}} \text{ task} \]

where task is the name of the task whose status is to be output.

The status-output for a ;TSTAT key-in request is of the form

\[ \text{task Level Status Tmmin TSmilli} \]

where

- task is the name of the task whose status is being output.
- level is the priority level (from 2 to 31) of the task.
- status is the status of the task as found in words 1 and 2 of the TIDB (Table 17-2).
- min is the value of the counter in TIDB word 11.
- milli is the value of the counter in TIDB word 10.

The values of min and milli are printed only if bit 0 and/or 7 of TIDB word 1 (Table 17-2) is set.

**Table 17-2. Task Status (TIDB Words 1 and 2)**

<table>
<thead>
<tr>
<th>TIDB Word</th>
<th>Bit</th>
<th>Meaning of Set Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td>Suspend interrupt</td>
</tr>
<tr>
<td>1</td>
<td>14</td>
<td>Suspend task</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>Abort task</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>Exit from task</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>TIDB resident</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>Resident task</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>Foreground task</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>Protected task</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>Task scheduled by time-delay</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>Time-delay active</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>Task waiting to be loaded</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>Task error</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>Task interrupt expected</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>Overlay task</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Scheduled task upon termination of active task</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Task search-allocated-loaded</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>Task opened</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>Task loaded in background (checkpoint) area</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>Load overlay</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>Unused</td>
</tr>
</tbody>
</table>

**Operator key-in examples:** Request the output of the status of the task BIGJOB.

\[ ; \text{\texttt{TSTAT,BIGJOB}} \]

The output will be

\[ \text{BIGJOB P02 0001000 000000 TM077777 T077430} \]

if the status of BIGJOB is such that it is on priority level 2, contains a status of 0100 in TIDB words 1 and 2, with time counters (TIDB words 10 and 11) of 077777 and 077430, respectively. The latter two octal complement counters show zero minutes and 0340.5 millisecond increments.

Request the output of the status of all foreground tasks inputs.

\[ ; \text{\texttt{TSTAT}} \]

and receive as a typical response

\[ \text{V2DB} P24 0047041 000000 TM077311 T0771000 \]
\[ \text{V6TMA} P23 0047044 000000 TM077600 T0771000 \]
\[ \text{VCLPA} P22 0047041 000000 TM077700 T0772200 \]
\[ \text{VCRRA} P22 0047041 000000 TM077700 T0770221 \]
\[ \text{VIMTA} P22 0047041 000000 TM077200 T0771000 \]
\[ \text{VIMTA} P22 0047041 000000 TM077200 T0771000 \]
\[ \text{V50PCM} P10 0056405 000000 TM077700 T0770333 \]
\[ \text{JCP} P01 0044000 000000 TM077000 T0770000 \]

**17.2.9 ;ASSIGN Key-In Request**

This key-in request equates and assigns particular logical units to specific I/O devices. It has the general form

\[ ; \text{\texttt{ASSIGN,(1)=r(1),...,(n)=r(n)}} \]

where

- each l(n) is a logical-unit number (e.g., 12) or name (e.g., SI).
- each r(n) is a logical-unit number or name, or a physical-device system name (e.g., TY00 or TY, Table 15-1).

The logical unit to the left of the equal sign in each pair is assigned to the unit/device to the right.

An inoperable device, i.e., one declared down by ;DEVDN (section 17.2.10), cannot be assigned. A logical unit designated as unassignable (unit numbers 101 through 179) cannot be reassigned.

**Operator key-in examples:** Assign the card reader CR00 as the SI logical unit and the Teletype TY01 as the OC unit.

\[ ; \text{\texttt{ASSIGN,SI=CR00,OC=TY01}} \]

Assign a dummy device as the PI unit.

\[ ; \text{\texttt{ASSIGN,PI=DUM}} \]
17.2.10 ;DEVDN (Device Down) Key-In Request

This key-in request declares the specified physical device inoperable for system use. It is not applicable to the OC unit or to devices containing system libraries. The request has the general form

;DEVDN,device

where device is the system name of the physical device in four ASCII characters, e.g., LP00 (or LP). TY01, (table 15-1)

Operator key-in example: Declare TY01 inoperable for system use.

;DEVDN,TY01

17.2.11 ;DEVUP (Device Up) Key-In Request

This key-in request declares the specified physical device operational for system use. It has the general form

;DEVUP,device

where device is the system name of the physical device in four ASCII characters, e.g., LP00 (or LP). TY01 (table 15-1)

Operator key-in example: Declare TY02 operational for system use.

;DEVUP,TY02

17.2.12 ;IOLIST (List I/O Key-In Request

This key-in request outputs a listing of the specified logical-unit assignments, if any. If no logical unit is specified, ;IOLIST outputs all logical-unit assignments. The key-in request has the general form

;IOLIST,lun(1),lun(2),...,lun(n)

where each lun(n) is the name or number of a logical unit, e.g., SI,5.

Where the ;IOLIST key-in request specifies a logical-unit name, the output is of the form

name (number) = device D

where

device is the name of the physical device assigned, e.g., LP00

D if present, indicates that the physical device has been declared down and is thus inoperable

If the key-in request specifies the number rather than the name of the logical unit, the output will repeat the number in both the name and number fields.

In a listing of all assignments, the output uses a name and number where applicable, and the repeated number where no name is assigned to the logical unit. Logical units without names assigned at system-generation time are not listed and must be individually specified by number.

Operator key-in examples: Request the output of the logical-unit assignments for the BI and BO units. Input

;IOLIST,BI,BO

and receive as a typical response

BI (006) = CRC0
BO (007) = CPC0 D

Request the output of the logical-unit assignment for logical unit 180. Input

;IOLIST,180

and receive as a typical response

180 (180) = D11H

Request the output of all logical-unit assignments. Input

;IOLIST

and receive as a typical response

OC (001) = TY00
SI (002) = TY00
SO (003) = TY00
PX (004) = CR00 D
LO (005) = LP00
BI (006) = CRC0 D
BO (007) = PT00
SS (008) = DO0H
PO (009) = DO0H
CU (100) = DO0E
GO (101) = DO03
SW (102) = DO0F
CL (103) = DO0A
OM (104) = DO0D
BL (105) = DO0C
FL (106) = DO03
19.4 ISA FORTRAN PROCESS CONTROL SUBROUTINES

The Instrument Society of America (ISA) has defined as standards a number of FORTRAN subprogram calls useful in process input/output applications. VORTEX includes the following subroutines of this group:

Input/Output Calls

- AISQ(W): Analog Input Sequential
- AIRD(W): Analog Input Random
- AO(W): Analog Output
- DI(W): Digital Input
- DOM(W): Digital Output Momentary
- DOL(W): Digital Output Latching

The (W) option with each of these subroutine names selects a 'wait' mode, that is, it specifies that return is not be made from the subroutine until the I/O is finished, either normally or erroneously.

Bit String Manipulation

- IOR: Inclusive OR (logical add)
- AND: AND (logical multiply)
- NOT: NOT (logical invert)
- IEOR: Exclusive OR (logical subtract)
- ISHFT: Logical Shift

VORTEX also provides two FORTRAN subprogram calls to set the amplifier gains on the Low-Level Multiplexors. The gain control calls are not ISA standard calls.

Low Level Gain Calls

- SGNF(D): Set gain on sequential channels
- SGNT(D): Set gains through a table

The (D) option of each of these routines cause a 5 millisecond delay after the last gain control has been issued, to give the amplifiers time to settle.

19.4.1 Input/Output Calls

The parameter 'stat' appears in all the following I/O calls. Its contents give the status of the call, as follows:

- stat = 1: I/O correctly completed
- 2: I/O in execution
- 3: Invalid channel number
- 4: BIC timeout error
- 5: Invalid parameter value
VORTEX provides a FORTRAN call which establishes execution-time association between channel numbers and logical unit numbers, and sets the timer for data input rate. The format is:

```
CALL V$OPIO (da, lun, time, stat)
```

where:
- **da** = device address
- **lun** = logical unit number
- **time** = time, in microseconds, between input. This is loaded into device programmable timer, which controls BIC rate. It is ignored on output. Parameters may be redefined by successive calls to V$OPIO.

### Read Analog Input Sequential

```
CALL AISQ (count, ptlist, ibuf, stat)
```

or

```
CALL AISQW (count, ptlist, ibuf, stat)
```

This call reads count analog inputs into buffer `ibuf`, starting with channel 0x001, where `ptlist` contains 0x0yy, and reading channels sequentially.

### Read Analog Input Random

```
CALL AIRD (count, ptlist, ibux, stat)
```

or

```
CALL AIRDW (count, ptlist, ibuf, stat)
```

This call reads count analog inputs into buffer `ibuf`, inputting from the list of random points `ptlist`.

### Perform Analog Output

```
CALL AO (count, ptlist, obuf, stat)
```

or

```
CALL AOW (count, ptlist, obuf, stat)
```

This call outputs count words of digital output from buffer `obuf`, outputting from the list of random digital channels `ptlist`. The device driver program will save the previous word output to each channel, and change only those bits specified by 1-bits in `mask` which is an integer array parallel to `obuf` and `ptlist`.

### Perform Digital Output - Momentary

```
CALL DOM (count, ptlist, obuf, time, stat)
```

or

```
CALL DOW (count, ptlist, obuf, time, stat)
```

This call outputs count words of digital output from buffer `obuf`, outputting from the list of random digital channels `ptlist`. If `time` = 0 this completes the operation. Otherwise, after `5*time` in milliseconds a word of zeros will be output to every channel in `ptlist`, thus resetting all channels.

### Perform Digital Output - Latching

```
CALL DOL (count, ptlist, obuf, mask, stat)
```

or

```
CALL DOLW (count, ptlist, obuf, mask, stat)
```

This call outputs count words of digital output from buffer `obuf`, outputting from the list of random digital channels `ptlist`. The device driver program will save the previous word output to each channel, and change only those bits specified by 1-bits in `mask` which is an integer array parallel to `obuf` and `ptlist`.

### Perform Gain Selection or Sequential Channels

```
CALL SGNF (chntbl, nochnl)
```

or

```
CALL SGNFD (chntbl, nochnl)
```

This call selects the gain on `nochnl` sequential low level input channels. Chntbl is the name of a two word control table. The first word contains the address of the first low level channel. The second word contains the gain parameter (0-7).

### Perform Gain Selection on Channels through a Table

```
CALL SGNT (chntbl, nochnl)
```

or

```
CALL SGNMTD (chntbl, nochnl)
```

197
This call selects gains on nochrl low level channels. Chutbl is the name of a table which contains a pair of words for control for each low level channel. The first word of each pair contains the address of the low level channel. The second word of each pair contains the gain parameter (0-7).

19.4.2 Bit String Operations

All these subprograms are defined as Integer Function Subprograms. In the following descriptions, m and n are integer mode expressions.

\[
\begin{align*}
\text{IOR}(m, n) &= m\lor n \\
\text{IAND}(m, n) &= m\land n \\
\text{NOT}(m) &= \neg m \\
\text{IEOR}(m, n) &= n\oplus m \\
\text{ISHFT}(m, n) &= 0 \\
\end{align*}
\]

- Inclusive OR (logical sum) AND (logical product)
- NOT (logical invert)
- Exclusive OR (logical difference)
- If the absolute value of \(m \geq 16\)

19.5 ERRORS

Process Output

IO03 INVALID CHANNEL NUMBER

Process Input

IO03 INVALID CHANNEL NUMBER
IO2X BIC TIMEOUT ERROR

19.6 EXTENSIONS

Other process control devices besides those in the table of section 19.1 may be brought into the VORTEX system at some future time. The procedure for entering a new process control device is as given for the currently supported devices: one codes a driver program and controller tables and enters them into the VORTEX Nucleus at SGEN time, remembering to increment the one-character suffix on all names (all names herein end in 'A'; the next type of DAC, say, would be tagged with 'B'). The controller table can be extended by as many words as desired, to store flags and fixed device parameters. For variable parameters, say a gain parameter on an analog input device, the PCB table can be extended to hold the new parameter. In the FORTRAN I/O calls, the array PTLIST can be made 2-dimensional if gain or other parameter information is to be transferred with each point or channel number.
This manual explains the Varian Omnitask Real-Time Executive (VORTEX) and its use, but it is not intended for a beginning audience. Prerequisite to an understanding of this manual is a knowledge of general programming concepts, and preferably some Varian Data Machines 620 series or V70 series computer system is desirable.
**APPENDIX A**

**ERROR MESSAGES**

This appendix comprises a directory of VORTEX operating system error messages, arranged by VORTEX component. For easy reference, the number of the subsection containing the error messages for a component ends with a number corresponding to that of the section that covers the component itself, e.g., the file-maintenance error messages are listed in subsection A.9 because the file-maintenance component itself is discussed in section 9.

### A.1 ERROR MESSAGE INDEX

Except for the language processors (section 5), VORTEX error messages each begin with two letters that indicate the corresponding component:

<table>
<thead>
<tr>
<th>Messages beginning with:</th>
<th>Are from component:</th>
<th>Listed in subsections:</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM</td>
<td>Concordance program</td>
<td>A.5</td>
</tr>
<tr>
<td>DG</td>
<td>Debugging program</td>
<td>A.7</td>
</tr>
<tr>
<td>DP</td>
<td>Dataplot II</td>
<td>A.12</td>
</tr>
<tr>
<td>EX</td>
<td>Real-time executive</td>
<td>A.2</td>
</tr>
<tr>
<td>FM</td>
<td>File maintenance</td>
<td>A.9</td>
</tr>
<tr>
<td>IO</td>
<td>I/O control</td>
<td>A.3</td>
</tr>
<tr>
<td>IU</td>
<td>I/O utility</td>
<td>A.10</td>
</tr>
<tr>
<td>JC</td>
<td>Job-control processor</td>
<td>A.4</td>
</tr>
<tr>
<td>LG</td>
<td>Load-module generator</td>
<td>A.6</td>
</tr>
<tr>
<td>MS</td>
<td>Microprogram simulator</td>
<td>A.18.1</td>
</tr>
<tr>
<td>MU</td>
<td>Microprogram utility</td>
<td>A.18.2</td>
</tr>
<tr>
<td>NC</td>
<td>VTAM Network control</td>
<td>A.20</td>
</tr>
<tr>
<td>OC</td>
<td>Operator communication</td>
<td>A.17</td>
</tr>
<tr>
<td>RP</td>
<td>RPG IV Compiler</td>
<td>A.3</td>
</tr>
<tr>
<td>RT</td>
<td>RPG IV Runtime/Loader</td>
<td>A.5.3</td>
</tr>
<tr>
<td>SE</td>
<td>Source editor</td>
<td>A.8</td>
</tr>
<tr>
<td>SG</td>
<td>System generator</td>
<td>A.15</td>
</tr>
<tr>
<td>SM</td>
<td>System maintenance</td>
<td>A.16</td>
</tr>
<tr>
<td>ST</td>
<td>VSORT</td>
<td>A.11</td>
</tr>
<tr>
<td>DAS MR assembler</td>
<td></td>
<td>A.5</td>
</tr>
</tbody>
</table>

Section A.21 gives explanations of error codes listed under "Possible User Action" in the last column of the following sections.

#### A.2 REAL-TIME EXECUTIVE

<table>
<thead>
<tr>
<th>Message</th>
<th>Condition</th>
<th>Action</th>
<th>Possible User Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>EX01,xxxxxx</td>
<td>Invalid RTE service request by task xxxxxx</td>
<td>Abort task xxxxxx</td>
<td>D01,D02,P01</td>
</tr>
<tr>
<td>EX02,xxxxxx</td>
<td>Scheduled task xxxxxx name not in specified load-module library</td>
<td>Abort task xxxxxx</td>
<td>D01,D03</td>
</tr>
<tr>
<td>EX03,xxxxxx</td>
<td>Task xxxxxx made RESUME request but requested task not found</td>
<td>Continue scheduling task</td>
<td>D01,D03</td>
</tr>
<tr>
<td>EX04,xxxxxx</td>
<td>Task xxxxxx made ABORT request but requested task not found</td>
<td>Task xxxxxx continues</td>
<td>D01,D03</td>
</tr>
<tr>
<td>EX05,xxxxxx</td>
<td>Background task xxxxxx larger than allocatable</td>
<td>Task xxxxxx not loaded</td>
<td>M01,M02,M03,M04,P02</td>
</tr>
<tr>
<td>EX06,xxxxxx</td>
<td>Not enough allocatable space available for ALOC request</td>
<td>Abort task xxxxxx</td>
<td>M01,M02,M03,M04</td>
</tr>
<tr>
<td>EX07,xxxxxx</td>
<td>OVLAY requests a segment not in library</td>
<td>Abort task xxxxxx</td>
<td>D01,D03</td>
</tr>
</tbody>
</table>
### ERROR MESSAGES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Action</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>EX10,xxxxx</td>
<td>Scheduled request has a library task priority conflict (task priority 0 from foreground library, task priority 2 from background library). Scheduled request specifies a foreground task to be executed at priority 0 or 1</td>
<td>Schedule request ignored, scheduling task continues</td>
<td>D04,D02,P01</td>
</tr>
<tr>
<td>EX11,xxxxx,n</td>
<td>Memory protection violation at address n</td>
<td>Abort task xxxxx</td>
<td>P03</td>
</tr>
<tr>
<td>EX12,xxxxx</td>
<td>I/O link error (foreground task making request, or incorrect logical unit number)</td>
<td>Abort task xxxxx</td>
<td>PC1</td>
</tr>
<tr>
<td>EX15,xxxxx</td>
<td>Foreground common specified by background task</td>
<td>Abort task xxxxx</td>
<td>PC1</td>
</tr>
<tr>
<td>EX16,xxxxx</td>
<td>PASS macro specified zero or negative word count</td>
<td>Abort task xxxxx</td>
<td>P01</td>
</tr>
<tr>
<td>EX17,xxxxx</td>
<td>RMD I/O error detected when SAL attempted to load scheduled task, xxxxxx. Also pseudo TIDB data assumed bad, execution address less than 01000</td>
<td>Abort task xxxxx</td>
<td>M06,P01</td>
</tr>
<tr>
<td>EX32,xxxxx</td>
<td>Attempted to schedule a task from a non-RMD unit</td>
<td>Directive ignored</td>
<td>D02,P01</td>
</tr>
<tr>
<td>EX33,xxxxx</td>
<td>Floating-point processor fault, FPP, error</td>
<td>Program continues at the address following the FPP instruction</td>
<td>None</td>
</tr>
<tr>
<td>EX34,xxxxx</td>
<td>Floating-point processor timeout</td>
<td>Program continues None at interrupted instruction</td>
<td></td>
</tr>
</tbody>
</table>

Note: xxxxx is the name of a task.
## A.3 I/O CONTROL

<table>
<thead>
<tr>
<th>Message</th>
<th>Condition</th>
<th>Action</th>
<th>Possible User Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>I000,xxxxx</td>
<td>Unit not ready, or unit file protected</td>
<td>Repeats message until condition is corrected</td>
<td>H01,H03</td>
</tr>
<tr>
<td>I001,xxxxx</td>
<td>Device declared down</td>
<td>Repeats message until condition is corrected</td>
<td>H04,D19</td>
</tr>
<tr>
<td>I002,xxxxx</td>
<td>Invalid LUN specified</td>
<td>Abort task or request</td>
<td>D02,P01</td>
</tr>
<tr>
<td>I003,xxxxx</td>
<td>FDB/DCB parameter error</td>
<td>Abort task or request</td>
<td>P04</td>
</tr>
<tr>
<td>I004,xxxxx</td>
<td>Invalid protection code</td>
<td>Abort task or request</td>
<td>D01,D02,P01</td>
</tr>
<tr>
<td>I005,xxxxx</td>
<td>Protected partition specified by unprotected task</td>
<td>Abort task or request</td>
<td>P01</td>
</tr>
<tr>
<td>I006,xxxxx</td>
<td>I/O request error, e.g., I/O-complete bit not set, prior request may be queued</td>
<td>Abort task or request</td>
<td>H05</td>
</tr>
<tr>
<td>I007,xxxxx</td>
<td>Attempt to read from a write-only device, or vice versa</td>
<td>Abort task or request</td>
<td>D02,P01</td>
</tr>
<tr>
<td>I010,xxxxx</td>
<td>File name specified in OPEN or CLOSE not found</td>
<td>Abort task or request</td>
<td>D01,D03,P01,D29</td>
</tr>
<tr>
<td>I011,xxxxx</td>
<td>Invalid file extent, record number, address or skip parameter</td>
<td>Abort task or request</td>
<td>P04,P01</td>
</tr>
<tr>
<td>I012,xxxxx</td>
<td>RMD OPEN/CLOSE error, or bad directory thread</td>
<td>Abort task or request</td>
<td>H05,D03</td>
</tr>
<tr>
<td>I013,xxxxx</td>
<td>Level 0 program read a JCP (/) directive</td>
<td>Task xxxx is aborted, directive passed to JCP buffer</td>
<td>None</td>
</tr>
<tr>
<td>I014,xxxxx</td>
<td>Interrupt timed out or no cylinder-search-complete interrupt</td>
<td>Abort task or request</td>
<td>H05,D05</td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
<td>Action</td>
<td>Code</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------------------------------</td>
<td>-----------------</td>
<td>-------</td>
</tr>
<tr>
<td>1015</td>
<td>Disc cylinder-search or malfunction error</td>
<td>Abort task or request</td>
<td>H05</td>
</tr>
<tr>
<td>1016</td>
<td>Disc read/write timing error</td>
<td>Abort task or request</td>
<td>H05</td>
</tr>
<tr>
<td>1017</td>
<td>Disc end-of-track error</td>
<td>Abort task or request</td>
<td>H05</td>
</tr>
<tr>
<td>1020</td>
<td>BIC1: abnormal stop, not ready, or time out error</td>
<td>Abort task or request</td>
<td>D05,H05</td>
</tr>
<tr>
<td>1021</td>
<td>BIC2: abnormal stop, not ready, or time out error</td>
<td>Abort task or request</td>
<td>D05,H05</td>
</tr>
<tr>
<td>1022</td>
<td>BIC3: abnormal stop, not ready, or time out error</td>
<td>Abort task or request</td>
<td>D05,H05</td>
</tr>
<tr>
<td>1023</td>
<td>BIC4: abnormal stop, not ready, or time out error</td>
<td>Abort task or request</td>
<td>D05,H05</td>
</tr>
<tr>
<td>1024</td>
<td>BIC5: abnormal stop, not ready, or time out error</td>
<td>Abort task or request</td>
<td>D05,H05</td>
</tr>
<tr>
<td>1025</td>
<td>BIC6: abnormal stop, not ready, or time out error</td>
<td>Abort task or request</td>
<td>D05,H05</td>
</tr>
<tr>
<td>1026</td>
<td>BIC7: abnormal stop, not ready, or time out error</td>
<td>Abort task or request</td>
<td>D05,H05</td>
</tr>
<tr>
<td>1027</td>
<td>BIC8: abnormal stop, not ready, or time out error</td>
<td>Abort task or request</td>
<td>D05,H05</td>
</tr>
<tr>
<td>1030</td>
<td>Parity error</td>
<td>Abort task or request</td>
<td>H05,D02</td>
</tr>
<tr>
<td>1031</td>
<td>Reader or tape error</td>
<td>Abort task or request</td>
<td>H05,P19</td>
</tr>
<tr>
<td>1032</td>
<td>Odd-length record error</td>
<td>Abort task or request</td>
<td>H05,P12</td>
</tr>
<tr>
<td>1033</td>
<td>Invalid terminal identifier or logical line number</td>
<td>Request ignored</td>
<td>D27</td>
</tr>
<tr>
<td>1034</td>
<td>Line or terminal not opened</td>
<td>Request ignored</td>
<td>D28</td>
</tr>
<tr>
<td>1035</td>
<td>Line or terminal down</td>
<td>Request ignored</td>
<td>D28</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td>Action</td>
<td>Reference</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------</td>
<td>----------------</td>
<td>------------</td>
</tr>
<tr>
<td>1036</td>
<td>Line or terminal already open</td>
<td>Request ignored</td>
<td>D28</td>
</tr>
<tr>
<td>1037</td>
<td>Request still pending</td>
<td>Request ignored</td>
<td>None</td>
</tr>
<tr>
<td>1040</td>
<td>Action on terminal not opened</td>
<td>Request ignored</td>
<td>D28</td>
</tr>
<tr>
<td>1042</td>
<td>Invalid physical line address</td>
<td>Request ignored</td>
<td>D27</td>
</tr>
<tr>
<td>1043</td>
<td>Invalid TCM type</td>
<td>Request ignored</td>
<td>D27</td>
</tr>
<tr>
<td>1044</td>
<td>No temporary storage available</td>
<td>Request ignored</td>
<td>None</td>
</tr>
<tr>
<td>1045</td>
<td>RMD error. Format, end-of-file or head selection error</td>
<td>Abort task or request</td>
<td>H05, D13</td>
</tr>
<tr>
<td>1047</td>
<td>User write specified word count &gt;73</td>
<td>Record is truncated</td>
<td>P04</td>
</tr>
<tr>
<td>105x</td>
<td>RMD read error on stream X, specified last digit of error number</td>
<td>The data is used</td>
<td>H06</td>
</tr>
<tr>
<td>1060</td>
<td>RMD file full</td>
<td>The program waits until space is available on the file. The message is repeated every 200 times the condition occurs</td>
<td>D08</td>
</tr>
<tr>
<td>1061</td>
<td>User parameter error in request</td>
<td>Request is ignored</td>
<td>P01</td>
</tr>
<tr>
<td>1062</td>
<td>RMD write error</td>
<td>The bad sector is skipped. This is likely to cause an 105x error later, but no data will be lost</td>
<td>H06</td>
</tr>
<tr>
<td>1063</td>
<td>Buffer unavailable for spooler</td>
<td>Spooler waits until buffer is available</td>
<td>None</td>
</tr>
</tbody>
</table>

Note: xxxxxx is the name of a task or device.
## A.4 JOB-CONTROL PROCESSOR

<table>
<thead>
<tr>
<th>Message</th>
<th>Condition</th>
<th>Action</th>
<th>Possible User Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>JC01</td>
<td>Invalid JCP directive</td>
<td>Ignore directive</td>
<td>D01, D02</td>
</tr>
<tr>
<td>JC02</td>
<td>Invalid or missing parameter in a JCP directive; or illegal separator or terminator</td>
<td>Ignore directive</td>
<td>D01, D02</td>
</tr>
<tr>
<td>JC03</td>
<td>Specified physical device cannot perform the functions of the assigned logical unit</td>
<td>Ignore directive</td>
<td>D07, H06</td>
</tr>
<tr>
<td>JC04</td>
<td>Invalid protection code or file name in a JCP directive</td>
<td>Ignore directive</td>
<td>D01, D02</td>
</tr>
<tr>
<td>JC05,nn</td>
<td>End of tape before the number of files specified by an /SFIL directive has been skipped; or end of tape, beginning of tape, or file mark before the number of records specified by an /SREC directive has been skipped where nn is the number of files (or records) remaining to be skipped</td>
<td>SFIL, SREC terminates upon error condition</td>
<td>P07</td>
</tr>
<tr>
<td>JC06</td>
<td>An irrecoverable I/O error while compiling or assembling; or an error during a load/go operation; or insufficient symbol table memory (insufficient /MEM directive), or an EOF was encountered before an END statement</td>
<td>Job flushed to next /JOB directive</td>
<td>P07, M01, P06</td>
</tr>
<tr>
<td>JC07</td>
<td>Invalid or illegal logical/physical-unit referenced in JCP directive</td>
<td>Ignore directive</td>
<td>D01, D02, H06</td>
</tr>
</tbody>
</table>

## A.5 LANGUAGE PROCESSORS

### A.5.1 DAS MR Assembler
A.5.2 FORTRAN IV Compiler and Runtime Compiler

During compilation, source statements are checked for such items as validity, syntax, and usage. When an error is detected, it is posted on the LO usually beneath the source statement. The errors marked T terminate binary output.

All error messages are of the form

```
ERR xx c(1)-c(16)
```

where xx is a number form 0 to 18 (notification error), or T followed by a number from 0 to 9 (terminating error); and c(1)-c(16) is the last character string (up to 16) encountered in the statement being processed. The right-most character indicates the point of error and the @ indicates the end of the statement. The possible error messages are:

<table>
<thead>
<tr>
<th>Notification Error</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Illegal character input</td>
</tr>
<tr>
<td>1</td>
<td>Construction error</td>
</tr>
<tr>
<td>2</td>
<td>Usage error</td>
</tr>
<tr>
<td>3</td>
<td>Mode error</td>
</tr>
<tr>
<td>4</td>
<td>Illegal DO termination</td>
</tr>
<tr>
<td>5</td>
<td>Improper statement number</td>
</tr>
<tr>
<td>6</td>
<td>Common base lowered</td>
</tr>
<tr>
<td>7</td>
<td>Illegal equivalence group</td>
</tr>
<tr>
<td>8</td>
<td>Reference to nonexecutable statement</td>
</tr>
<tr>
<td>9</td>
<td>No path to this statement</td>
</tr>
<tr>
<td>10</td>
<td>Multiply defined statement number</td>
</tr>
<tr>
<td>11</td>
<td>Invalid format construction</td>
</tr>
<tr>
<td>12</td>
<td>Spelling error</td>
</tr>
<tr>
<td>13</td>
<td>Format statement with no statement number</td>
</tr>
<tr>
<td>14</td>
<td>Function not used as variable</td>
</tr>
<tr>
<td>15</td>
<td>Truncated value</td>
</tr>
<tr>
<td>16</td>
<td>Statement out of order</td>
</tr>
<tr>
<td>17</td>
<td>More than 29 named common regions</td>
</tr>
<tr>
<td>18</td>
<td>Noncommon data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Terminating Error</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>I/O error</td>
</tr>
<tr>
<td>T1</td>
<td>Construction error</td>
</tr>
<tr>
<td>T2</td>
<td>Usage error</td>
</tr>
<tr>
<td>T3</td>
<td>Data pool overflow</td>
</tr>
<tr>
<td>T4</td>
<td>Illegal statement</td>
</tr>
<tr>
<td>T5</td>
<td>Improper use</td>
</tr>
<tr>
<td>T6</td>
<td>Improper statement number</td>
</tr>
<tr>
<td>T7</td>
<td>Mode error</td>
</tr>
<tr>
<td>T8</td>
<td>Constant too large</td>
</tr>
<tr>
<td>T9</td>
<td>Improper DO nesting</td>
</tr>
</tbody>
</table>

Note: due to optimization, the error message may appear on the next labeled statement and not on the actual statement error.

RUNTIME

When an error is detected during runtime execution of a program, a message is posted on the LO device of the form:

```
taskname message
```

Fatal errors cause the job to be aborted; execution continues for non-fatal errors. The messages and their definitions are:

<table>
<thead>
<tr>
<th>Message</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARITH OVFL</td>
<td>Arithmetic overflow</td>
</tr>
<tr>
<td>GO TO RANGE</td>
<td>Computed GO TO out of range*</td>
</tr>
<tr>
<td>FUNC ARG</td>
<td>Invalid function argument (e.g., square root of</td>
</tr>
<tr>
<td></td>
<td>negative number)</td>
</tr>
<tr>
<td>FORMAT</td>
<td>Error in FORMAT statement*</td>
</tr>
<tr>
<td>MODE</td>
<td>Mode error (e.g., outputting real array with I format)*</td>
</tr>
<tr>
<td>DATA</td>
<td>Invalid input data (e.g., inputting a real number</td>
</tr>
<tr>
<td></td>
<td>from external medium with I format)*</td>
</tr>
<tr>
<td>I/O</td>
<td>I/O error (e.g., parity, EOF)*</td>
</tr>
</tbody>
</table>

* indicates fatal error; all others non-fatal

A.5.3 RPG IV Compiler and Runtime Compiler

During compilation, source statements are checked for such items as validity, syntax and usage. When an error is detected an arrow is printed pointing to the discrepancy in the source statement and an error message is output on the LO device. Detailed descriptions can be found in the RPG IV User's Manual (98 A 9947 03X). The possible error messages are:

```
Messages
```

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid</td>
<td>Relational</td>
</tr>
<tr>
<td>Label</td>
<td>Size</td>
</tr>
<tr>
<td>Literal</td>
<td>Syntax</td>
</tr>
</tbody>
</table>

If an I/O error occurs during compilation one of the following messages is posted on Logical Unit 15 and compilation is terminated:
### A.6 LOAD-MODULE GENERATOR

<table>
<thead>
<tr>
<th>Message</th>
<th>Condition</th>
<th>Action</th>
<th>Possible User Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>LG01</td>
<td>Invalid LMGEN directive</td>
<td>Ignore directive</td>
<td>D01,D02</td>
</tr>
<tr>
<td>LG02</td>
<td>Invalid or missing parameter in an LGMEN directive</td>
<td>Ignore directive</td>
<td>D01,D02</td>
</tr>
<tr>
<td>LG03</td>
<td>Check-sum error in object module</td>
<td>Abort loading</td>
<td>P08,P09</td>
</tr>
<tr>
<td>LG04</td>
<td>READ error in object module</td>
<td>Abort loading</td>
<td>P08,H06</td>
</tr>
<tr>
<td>LG05</td>
<td>WRITE error in load loading</td>
<td>Abort loading</td>
<td>P08,H06</td>
</tr>
<tr>
<td>LG06</td>
<td>Cataloging error, name already in library, library full</td>
<td>Abort loading</td>
<td>D03,H06</td>
</tr>
<tr>
<td>LG07</td>
<td>Loader code error in object module</td>
<td>Abort loading</td>
<td>P08</td>
</tr>
<tr>
<td>LG08</td>
<td>Sequence error in object module</td>
<td>Abort loading</td>
<td>P08</td>
</tr>
<tr>
<td>LG09</td>
<td>Structure error in object module</td>
<td>Abort loading</td>
<td>P08</td>
</tr>
<tr>
<td>LG10</td>
<td>Literal pool overflow or use of literal by foreground program</td>
<td>Abort loading</td>
<td>P08,P09</td>
</tr>
<tr>
<td>LG11</td>
<td>Invalid redefinition of common-block size during load-module generation</td>
<td>Abort loading</td>
<td>P08</td>
</tr>
<tr>
<td>LG12</td>
<td>Load-module size exceeds available memory</td>
<td>Abort loading</td>
<td>P02</td>
</tr>
<tr>
<td>LG13</td>
<td>LMGEN internal tables exceed available memory</td>
<td>Abort loading</td>
<td>M01</td>
</tr>
</tbody>
</table>
### LG14
Number of overlay segments input not equal to that specified in TIDB
Abort loading D01,D02

### LG15
Undefined externals
Loading continues P10

### LG16
No program execution address
Loading continues. Address defaults to the first location of the program P17

### LG17
Attempt to load protected task on background library or unprotected task on foreground library
Abort loading D01,D02

### A.7 DEBUGGING PROGRAM

<table>
<thead>
<tr>
<th>Message</th>
<th>Condition</th>
<th>Action</th>
<th>Possible User Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>DG01</td>
<td>Invalid DEBUG directive</td>
<td>Ignore directive</td>
<td>D01,D02</td>
</tr>
<tr>
<td>DG02</td>
<td>Invalid or undefined parameter in DEBUG directive</td>
<td>Ignore directive</td>
<td>D01,D02</td>
</tr>
</tbody>
</table>

### A.8 SOURCE EDITOR

<table>
<thead>
<tr>
<th>Message</th>
<th>Condition</th>
<th>Action</th>
<th>Possible User Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>SE01</td>
<td>Invalid SEDIT directive</td>
<td>Directive ignored</td>
<td>D01,D02</td>
</tr>
<tr>
<td>SE02</td>
<td>Invalid or missing parameter in SEDIT directive</td>
<td>Directive ignored</td>
<td>D01,D02</td>
</tr>
<tr>
<td>SE03</td>
<td>Error reported by IOC call</td>
<td>Edit terminated</td>
<td>H06</td>
</tr>
<tr>
<td>SE04</td>
<td>Invalid end of file</td>
<td>Edit terminated</td>
<td>P07</td>
</tr>
</tbody>
</table>
## A.9 FILE MAINTENANCE

<table>
<thead>
<tr>
<th>Message</th>
<th>Condition</th>
<th>Action</th>
<th>Possible User Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>FM01</td>
<td>Invalid FMAIN directive</td>
<td>Ignore directive</td>
<td>D01, D02</td>
</tr>
<tr>
<td>FM02</td>
<td>Name already in directory</td>
<td>Module not added</td>
<td>D03, D01, D02, D07</td>
</tr>
<tr>
<td>FM03</td>
<td>Name not in directory</td>
<td>Module not deleted</td>
<td>D03, D01, D02</td>
</tr>
<tr>
<td>FM04</td>
<td>Insufficient space for entry</td>
<td>Module not added</td>
<td>D07, D08, D09</td>
</tr>
<tr>
<td>FM05</td>
<td>I/O error</td>
<td>FMAIN process terminated</td>
<td>H06</td>
</tr>
<tr>
<td>FM06</td>
<td>Directory structure error, including writing over the directory by direct addressing of an RMD partition</td>
<td>FMAIN process terminated</td>
<td>H06</td>
</tr>
<tr>
<td>FM07</td>
<td>Check-sum error in object module</td>
<td>FMAIN process terminated</td>
<td>P08</td>
</tr>
<tr>
<td>FM08</td>
<td>No entry name in object module</td>
<td>FMAIN process terminated</td>
<td>P08</td>
</tr>
<tr>
<td>FM09</td>
<td>Record-size error in object module</td>
<td>FMAIN process terminated</td>
<td>P12</td>
</tr>
<tr>
<td>FM10</td>
<td>Loader code error in object module</td>
<td>FMAIN process terminated</td>
<td>P08</td>
</tr>
<tr>
<td>FM11</td>
<td>Sequence error in object module</td>
<td>FMAIN process terminated</td>
<td>P08</td>
</tr>
<tr>
<td>FM12</td>
<td>Non-binary record in object module</td>
<td>FMAIN process terminated</td>
<td>P12</td>
</tr>
<tr>
<td>FM13</td>
<td>Number of input logical unit not specified by INPUT</td>
<td>FMAIN process terminated</td>
<td>D01, D02</td>
</tr>
<tr>
<td>FM14</td>
<td>Insufficient space in memory</td>
<td>FMAIN process terminated</td>
<td>M01</td>
</tr>
</tbody>
</table>

* Messages FM07 through FM14 apply only to the processing of object modules. The occurrence of any of these errors requires that the processing of the object module be restarted after the error condition is removed.
### A.17 OPERATOR COMMUNICATION

<table>
<thead>
<tr>
<th>Message</th>
<th>Condition</th>
<th>Action</th>
<th>Possible User Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>OC01</td>
<td>Request type error</td>
<td>Ignore directive</td>
<td>D01, D02</td>
</tr>
<tr>
<td>OC02</td>
<td>Parameter limits exceeded</td>
<td>Ignore directive</td>
<td>D01, D02</td>
</tr>
<tr>
<td>OC03</td>
<td>Missing parameter</td>
<td>Ignore directive</td>
<td>D01, D02</td>
</tr>
<tr>
<td>OC04</td>
<td>Unknown or undefined parameter</td>
<td>Ignore directive</td>
<td>D01, D02</td>
</tr>
<tr>
<td>OC05</td>
<td>Attempt to schedule or time schedule OPCOM task</td>
<td>Ignore directive</td>
<td>D01, D02</td>
</tr>
<tr>
<td>OC06</td>
<td>Attempt to declare OC device or system resident unit down</td>
<td>Ignore directive</td>
<td>D01, D02</td>
</tr>
<tr>
<td>OC07</td>
<td>Task specified in TSTAT key-in has no established TIDB, task currently not active</td>
<td>Ignore directive</td>
<td>D01, D02</td>
</tr>
<tr>
<td>OC10</td>
<td>Attempt to assign unit declared down or assign an unassignable logical unit/device</td>
<td>Ignore directive</td>
<td>D19, H04</td>
</tr>
<tr>
<td>OC11</td>
<td>Attempt to allocate TIDB unsuccessful for TSCHED request</td>
<td>Ignore directive</td>
<td>M02</td>
</tr>
</tbody>
</table>

### A.18 RMD ANALYSIS AND INITIALIZATION

<table>
<thead>
<tr>
<th>Message</th>
<th>Condition</th>
<th>Action</th>
<th>Possible User Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>RZ01</td>
<td>Invalid RAZI directive or illegal separator or terminator</td>
<td>Ignore directive</td>
<td>D01, D11</td>
</tr>
<tr>
<td>RZ02</td>
<td>Invalid parameter in a RAZI directive</td>
<td>Ignore directive</td>
<td>D01, D11</td>
</tr>
<tr>
<td>RZ03</td>
<td>Insufficient or conflicting directive information</td>
<td>Ignore directive</td>
<td>D01, D11</td>
</tr>
<tr>
<td>RZ04</td>
<td>New PST incompatible with the system</td>
<td>Ignore directive</td>
<td>D20, D21, D22, D11</td>
</tr>
</tbody>
</table>
A.21 ERROR CODES

A.21.1 Errors Related to Directives

D01 Check spelling, delimiters, and parameters.
D02 Enter corrected request from OC or SO.
D03 Check specified library for module name (FMAIN list).
D04 Correct task priority.
D05 Check PIM directives used at system generation.
D06 Use a global logical unit in directive.
D07 Use an alternate library or unit.
D08 Increase library size with RAZI or during SGEN.
D09 Delete unused modules from library.
D10 Reposition record if PT or CR (for MT or RMD positioning) is automatic and enter on SO.

R@ to reread the record or
P@ to reread the program or
/SMAIN@ to restart SMAIN where @ is a carriage return

D11 Correct input record by entering it on SO or indicate that it is positioned for rereading by entering C on SO.
D12 Restart component by entering C on SO. (Repositioning is automatic for MT and RMD, for cards reload the entire deck and SYSGEN will find component.)
D14 Restart SGEN from beginning.
D15 Check spelling, delimiters, etc., of IO INTER-GATION.
D16 Correct appropriate SGEN directives as indicated.
D17 Correct indicated module for next SGEN or add corrected module with LMGEN after SGEN completes

D18 Check that all RMDs are included in the SYS directive that are indicated by the EQUIP directives.
D19 Use OPCOM IOLIST for unit to check unit status (up or down) and unit's logical group.
D20 Check PRT directive
D21 Check if maximum number of partitions specified in EDR directive has been exceeded.
D22 Check for conflicts in controller/unit relations.
D23 Check logical unit in directive, must be assigned to first partition of the subject RMD unit.
D24 The specified RMD pack cannot contain a bad track table due to the first track being bad, use another pack.
D25 Check FRM directive and total number of tracks specified in PRT directive. The following table gives the track capacity for the standard RMDs:

<table>
<thead>
<tr>
<th>Model</th>
<th>Tracks</th>
</tr>
</thead>
<tbody>
<tr>
<td>70-75XX</td>
<td>4060 tracks</td>
</tr>
<tr>
<td>70-76XX</td>
<td>203 tracks</td>
</tr>
<tr>
<td>70-7701</td>
<td>128 tracks</td>
</tr>
<tr>
<td>70-7702</td>
<td>256 tracks</td>
</tr>
<tr>
<td>70-7703</td>
<td>512 tracks</td>
</tr>
</tbody>
</table>

D26 Check response to the highest page number requested.
D27 Check NDM definition or use LIST directive of NCM.
D28 Use NCM module to check line/terminal status.
D29 Check that all subject logical units assigned to RMD have been positioned with a PFIL.
D30 Use a larger file for the plot file.
D31 Check for proper logical unit (i.e., IOLIST).
D32 Increase work file xxxxxx size.

A.21.2 Errors Related to Programs

P01 Correct request in requesting task and re-execute.
A.21.3 Errors Related to Memory Size

M01 If background, adjust MEM directive as needed.

M02 Wait for foreground tasks to release memory or TIDB space.

M03 If MEM request OK or cannot be increased then cut back on foreground common, empty TIDBs, retry stack size, peripheral drivers, etc. by re-SGEN.

M04 If sharing blank common and VTAM LCB area, check that a program has not used part of the LCB area.

M05 Increase buffer area with BSS or dimension commands.

A.21.4 Errors Related to Hardware

H01 Make indicated unit ready.

H02 Clear the protection of the unit. (Disc write protection or write ring in MT)

H03 ABORT task, reassign SI if necessary, and then declare device down through OPCOM, do not forget to declare it back up again.

H04 ABORT task and assign alternate device or declare device back up.

H05 Check hardware for indicate problem.

H06 Check the OC device for an IO error message, i.e., IOxx.
E1314  Chemical Engineering Computing

Assignment 3: Use of Foreground

Modify the graphics program (Assignment 2),

(i) to evaluate the mean deviation, (the sum of the absolute deviations of the data points from the values given by the correlation divided by the number of points), and to print out $A$, $x'$, $a$ and the mean deviation on a designated device.

(ii) to allow choice of the device for inputting the data and printing out the above values.

Compile the program and generate a load module in your disc area. Run as follows -

(a) schedule from the OC,

(b) answer requests on the OC for identification of input device (test with 27 the outside TTY).

(c) answer request on this device for output device (test with 25 the line printer)

(d) input data, and proceed with program for a few values of the parameters.

Submit listing of program, teletype printout and line printer output.

Due date: September 17th.

Assignment 4: Sensing and Switching

Write a program in FORTRAN to carry out the following sequencing problem. This problem requires only digital sensing and switching.

A plastic is to be made in a reactor according to the following procedure:

(i) fill the tank with reactants (monomer, solvent, catalyst)

(ii) turn on the stirrer and heater,

(iii) after 2 minutes when the polymerisation reaction is complete, empty the tank for the next cycle.

There are two major restrictions.

(a) the stirrer must not be used unless the tank is at least half full (otherwise it will vibrate dangerously)

(b) the heater must not be used unless the tank is full (otherwise it will burn out)

There are other obvious restrictions; for example, it would be unwise to try to fill the tank with the drain valve open.

The apparatus is set up in the laboratory using water instead of the reagents and with the drain valve positioned above the heater to prevent accidental burn out in testing. Electrical leads are identified as follows: 1 'feed valve, 2' drain valve, 3 'stirrer, 4 'heater, 5 midlevel, 6 high level. (5v).

Submit a listing of a program. Due date: September 24th
CALL Pulse (6) to on
CALL Pulse (5) to off

fill

is drain open?

yes

is heat off?

no

20 KE