ICL

1900 Driver

1900 Series

The policy of International Computers Limited is one of continuous development and improvement of its products and services, and the right is therefore reserved to alter the information contained in this document without notice. ICL makes every endeavour to ensure the accuracy of the contents of this document but does not accept liability for any error or omission. Any equipment or software performance figures and times stated herein are those which ICL expects to be achieved in normal circumstances. Wherever practicable, ICL is willing to verify upon request the accuracy of any specific matter contained in this document.

Technical Publication 4285

Preliminary Edition February 1972

Reprinted May 1972

(incorporating Amendment List 1)

Issued by Technical Publications Service
International Computers Limited
Head Office: ICL House, Putney, London SW15
Produced by ICL Printing Services

Preface

This manual is a guide to the design and implementation of real time transaction processing programs using 1900 Driver. A description of such programs, and the operation of Driver, may be found in the ICL 1900 Series manual *Introduction to 1900 Driver* (Edition 1, TP 4311).

The manual is for use by systems designers and programming staff. Chapter 1 gives a general description of the operation of Driver facilities. Chapter 2 describes the factors to be taken into account when designing Driver programs. Chapters 3, 4, 5 and 6 describe the programming standards to be observed in producing the user-written part of a Driver program. Advice is given on designing and writing individual routines, together with details of the options available. Chapter 7 describes a recommended procedure for testing user-written routines, whilst Chapter 8 comprises specifications of the programs and routines provided to carry out this task.

1900 Driver may be used on a 1900 Series central processor using a multi-programming overlaid Executive or GEORGE 3.

Contents

List of illustrations

Chapter 1

Introduction

GENERAL

1900 Driver is a set of routines for users wishing to implement real time transaction processing systems on a 1900 Series configuration. Two types of routines are provided: *control routines* and *testing aids*.

Control routines

Driver control routines are provided to carry out the complex administrative tasks required by real time programs during their operation. These tasks include scheduling and initiating all communication between the program and its environment (i.e. communications and peripheral input/output, and overlaying), supervising the processing of messages by user written routines, and, in some cases, making available areas of main store to meet the processing requirements of specific messages.

The control routines are consolidated into the user's program and remain resident in main store throughout its operation. Facilities are available for the user to omit, modify or replace certain control routines in order to achieve the best possible response times consistent with the amount of main store available to the program. Use of the routines thus enables the individual user to save a considerable amount of programming effort without losing ultimate control over the operation of his program.

A further feature of the control routines is that, subject to certain restrictions, they are suitable for use in conjunction with standard ICL housekeeping and overlay packages. If these

are used, the user-written part of the program may be restricted almost entirely vetting input data and using it to generate data for output; the only exceptions being application orientated control functions such as initialization, file handling, message recognition and error recovery. All other non-processing functions are initiated by passing control to the appropriate sequence of control routines.

Testing aids

The testing aids enable the real time environment created by the control routines to be realistically simulated under batch testing conditions. Routines provided by the user for incorporation into a Driver program may be tested individually or in groups before inclusion in the program. Use of the testing aids is fully described in Chapters 7 and 8.

STRUCTURE AND OPERATION OF A DRIVER PROGRAM

Program structure

A Driver program is composed of *modules*. These are routines written in such a way that each forms a logically complete st in the processing of one or more input messages. This modular structure enables individual routines to be written, tested or modified without reference to the remainder of the program, provided that they conform to the standards outlined later on in this chapter.

The modules forming a Driver program are organised into two groups, namely the *control element* and the *processing element*. The control element, otherwise known as the *Driver*, comprises a suitable selection of the 1900 Driver control routines; these routines interface with further items of stadard software and/or user written PLAN routines. The processing element

consists entirely of user written PLAN or COBOL routines known as *beads*.

The processing of an input message involves the execution of routines in both elements of the program. The sequence of routines which must be executed in order to process a particular message is known as that message's *thread*. A program which can only support the processing of one message at a time is known as a *single threading* program, whilst a program which can handle several different messages simultaneously is known as a *multithreading* program. Since multithreading operation is considerably more complex than single threading, two main versions of the 1900 Driver package are available, one for each type of operation.

Method of operation

BASIC METHOD

Under both single and multithreading versions of Driver, the method by which individual messages are processed is essentially the same. On input of a message to the program, Driver *allocates* to the message a parameter area of standard size and format known as a *Task Administration Block or TAB*. Allocation is performed by placing the message in a data area permanently chained to the TAB. Control is then passed to the first bead in the message's thread and processing of the message proceeds.

The beads forming the processing element cannot pass control between themselves: transfer of control between beads is a Driver function. Similarly, they may not perform any other functions of Driver, for example, store allocation or any form of input or output. When, in the course of processing a message, a bead reaches a point in its execution where a Driver function, otherwise known as a *Driver service*, is required, it sets parameters in the TAB specifying the action to be taken

and passes control to Driver via a standard entry point. This procedure is known as *issuing a request*. A message's thread therefore enters the Driver at each point where a bead in the thread issues a request for a Driver service. If a request specifies that control is to be passed to another bead, or that the bead issuing the request is to be re-entered in order to complete its execution, Driver passes control to the nominated bead and the thread hence re-enters the processing element.

If required, the less frequently used beads of a program may be overlaid. Where the overlay facilities available with Driver are included, Driver checks each request for transfer of control to a bead to determine whether the specified bead is in main store. If this is not the case, Driver brings the bead into main store and then passes control to it in the normal manner. This operation is entirely automatic: a bead issuing a request makes no distinction as to whether the next bead required is store resident or transient. For this reason, both types of bead are written to exactly the same standards.

In the course of its execution, a bead may need to access input data in the form of the input message and, possibly, records read in from backing store. Conversely, at least one of the beads in the message's thread will usually be required to generate a reply message, whilst other beads may be required to generate data for updating a file. In some cases, a bead may also need to pass data to the next bead in the current message's thread: this data must be stored in such a way that there is no risk of it being corrupted during the execution of the intermediate routines within Driver. To enable Driver to pass input data to the beads for processing, and, conversely, to allow the beads to store data for output by Driver, two data areas, known as the *Message area* and the *Input/Output area*, are chained to the TAB. During the processing of the input message to which the TAB is allocated, the Message area holds the input message and is also used by the beads to store reply messages prior to output. Similarly, any data transferred

between the program and backing store during the processing of the message is stored in the Input/Output area following input or before output. A third area, known as the *Additional Core area* may also be chained to the TAB: this area can be used for transfer of data between one bead and another, and also to provide additional working storage for both beads and user written control routines.

When the last bead in a message's thread reaches the end of its execution, its final request (typically for output of a reply message) also specifies that the TAB in use is to be *deallocated*. Driver performs deallocation by clearing the TAB of all variable parameters, ready for allocation to another input message.

ADDITIONAL FEATURES OF MULTITHREADING OPERATION

The basic principles of multithreading are exactly the same as those of multiprogramming, the difference being that time sharing occurs between message threads instead of between free-standing programs.

The processing of a message will evidently be delayed at each point where its thread passes through the Driver. For example, if a bead processing a message issues a request for peripheral input, processing of the massage cannot continue until Driver has obtained the required data and passed control to the next bead in the message's thread. Whenever the processing of a message is delayed in this way, the multithreading Driver uses the processor time and thus made available to initiate or complete the servicing of as many other requests as possible. This includes passing control to a selected bead to enable the processing of another message to begin or continue.

The simultaneous handling of several different messages in a single program imposes a fairly complex message scheduling task

in the multithreading Driver. For example, Driver must ensure that each message receives a reasonable share of processor time. If this is not done, there is a possibility of individual messages being locked out of the processing element, resulting in unacceptably delayed response times. To describe the same situation in another way, the processing delays experienced by individual messages due to the competing demands of other messages for processing and Driver servicing must be kept to a minimum. A further factor to be taken into account is that the delay incurred whilst Driver is servicing a particular request, for example accessing a particular file, means that further identical requests from beads in other threads may accumulate whilst the request is being serviced.

The multithreading Driver carries out message scheduling by means of a *TAB queuing system*. Under this system, Driver maintains one or more chained queues of TABs for each Driver service: for example, a separate queue of TABs is usually maintained for each peripheral device or peripheral control unit, depending on the configuration in use. A TAB containing a request for a particular service is placed on the appropriate queue following entry to Driver from the bead issuing the request.

Since each TAB in a multithreading program is allocated to a different message, the order in which inidvidual TABs are queued and their contained requests are serviced directly influences the response times of the messages to which the TABs are allocated. Under the standard multithreading Driver, requests are serviced in a *first in first out* basis: each TAB is placed at the end of a queue and servicing of its request cannot continue until the preceding TABs in the same queue have had their requests serviced.

The user may, if he wishes, implement a *message priority*

system allowing for especially rapid processing of selected message types. The standard queueing system can be modified so that TABs found to be allocated to high priority messages are queued in front of TABs allocated to lower priority messages. Facilities are also available for manipulating message priorities in the course of Driver servicing to avoid the possibility of low priority messages being locked out of the processing element.

The other major feature distinguishing multithreading from single threading operation is the facility for *dynamic store allocation*. Under this system, Input/Output and Additional Core areas need not be permanently dedicated to each TAB, but may instead be allocated from a *common pool* of *store cells* when requested by a bead. A bead requesting this service specifies the size(s) of the cell(s) required as Input/Output and/or Additional core areas during the processing of the current message: Driver accordingly chains a cell or cells of the required size to the TAB allocated to the message. On deallocation of the TAB from its message on completion of processing, the cells are returned to the common pool, ready for allocation to another TAB.

Compared with the conventional *fixed store* system, whereby Input/Output and Additional Core areas are permanently chained to each TAB, use of a properly organised dynamic store system can considerably reduce a program's store requirements. In a fixed store system, the chained areas must evidently be of sufficient size to cope with the store requirements of all message types: if the requirements of different messages vary widely, some areas of store will be largely redundant for a high proportion of the time that the program is operational. Moreover, since each Input/Output and Additional Core area is permanently dedicated to a particular TAB, ir cannot be used whilst its TAB is inactive, i.e. awaiting allocation to an input message. In a dynamic store system, by contrast, store cells of varying sizes are distributed between TABs according

to the processing requirements of the messages currently passing through the program: an individual cell becomes available for use by another TAB as soon as the TAB to which it is chained is deallocated. However, this relatively efficient method of store utilization involves some increase in overall response times, due to the additional entries to Driver each time store cells are allocated or deallocated. For this reason, a fixed store system is usually employed if there is sufficient main store available.

DRIVER ROUTINES

Control routines

A standard single or multithreading Driver may contain up to six control routines. These are

Request Analyser
System Control
Communications Monitor
Peripheral Monitor
Store Administrator
Bead Scheduler

The main functions of the control routines are shown below: detailed information is given in Chapters 5 and 6. In particular, some of the multithreading versions of the routines additionally perform complex queue handling functions and make extensive use of subroutines; these operations are fully described in Chapter 6.

Control routine name	*Main function*	*Other major functions*
Request Analyser	Analyses requests, in each case initiating the	1 Stores current contents of

Control routine name	*Main function*	*Other major functions*
	required action via System Control.	accumulators 0 and 4 to 7 in TAB on receipt of each request. 2 *Multithreading option* Initiates overlay via subroutine (optional).
System Control	Controls entry to Peripheral Monitor, Communications Monitor, Store Administrator and Bead Scheduler.	1 *Multithreading version only* Initiates general suspension of program when no further processing is possible due to lack of input messages.
Communications Monitor	Initiates servicing of communications input/ output requests by Communications housekeeping.	1 Allocates a TAB to each input message. 2(a) *Single threading version* Performs TAB deallocation. (b) *Multithreading version* Initiates TAB deallocation by subroutine.

Control routine name	*Main function*	*Other major functions*
		3 *Multithreading version only* Initiates TAB queueing by subroutines.
Peripheral Monitor	Initiates servicing of peripheral input/ output requests by Direct Access/ Magnetic Tape house-keeping and/or user written file handling routines.	1 *Multithreading version only* Initiates TAB deallocation by subroutine (if required). 2 *Multithreading version only* Initiates TAB queueing by subroutines.
Store Administrator	*Single threading version* Optional dummy routine. Not normally required, but see Chapter 5 page 34.	1 *Multithreading version only* Initiates TAB queueing by subroutines.

Control routine name	*Main function*	*Other major functions*
	Multithreading version Services store requests (dynamic store systems only)	
Bead Scheduler	Passes control to next bead required following either (a) A specific request for transfer of control to another bead, or	1(a) *Single threading version* Initiates overlay by subroutine (optional). (b) *Multithreading versions* Initiates overlay via subroutine (optional).
	(b) Servicing of any other type of request	2 *Multithreading versions only* Initiates TAB queueing by subroutines.

Housekeeping

As indicated above, standard housekeeping may be used in conjunction with Peripheral Monitor and Communications Monitor. However, in multithreading programs, any storage device housekeeping package used is run in program member 3, to enable processing to continue while the peripheral transfer is in progress.

File handling routines

If direct access or magnetic tape housekeeping is used in a multithreading program, Peripheral Monitor is only able to initiate or complete the servicing of one peripheral request in each entry. If, instead, peripheral requests for access to different files or devices are distributed between a number of user-written file handling routines coded at PERI level, the multithreading Peripheral Monitor is able to initiate or complete the servicing of several peripheral requests in a single pass, thus improving the program's throughput.

Use of separate file handling routines in a single threading implementation of Driver can considerably simplify the task of subsequently modifying the program for multithreading oper tion.

Where direct access or magnetic tape housekeeping is used, the user is still recommended to activate these packages via file handling routines. This preserves the modular structure of the program, with the usual advantages in terms of ease of programming and enhancement.

Overlay routines

Overlay in both versions of Driver is carried out by the standard ICL overlay routine %EROL.

In the single threading version of Driver, %EROL is entered directly from Bead Scheduler whenever the latter routine detects that the next bead to be entered must first be overlaid In the multithreading version, %EROL operates autonomously in a separate program number in order to avoid suspension of the main program occurring during overlay transfers. In this case a further subroutine, named *Overlay Control* is present with

Driver: this subroutine may be called by both Request Analyser and Bead Scheduler in order to initiate overlay by %EROL.

THE MASTER ROUTINE

The User must write a PLAN segment, known as the Master routine, to carry out initialization procedures and make the first entry to Driver at the beginning of the program run. The required coding is described in Chapter 4.

Chapter 2

Program Design & Implementation

INTRODUCTION

This chapter contains guidelines and recommendations to the programmer designing a Driver program, together with considerations to be taken into account during production of specifications and timetables. Detailed information on the various modules of a Driver program will be found in later chapters of this manual.

PLANNING THE PROGRAM STRUCTURE

Having decided that 1900 Driver will be used in the planned program, the basic structure of the program is fixed. This structure is as described in the manual *Introduction to 1900 Driver* and in Chapter 1 of this manual. During the specification stage, however, choices need to be made from further alternatives in order to create the most efficient program for the job. The various decisions to be made are described in the remainder of the chapter.

THE CONTROL ELEMENT

Single threading versus multithreading

The first decision is whether a single threading or a multithreading Driver should be used. This decision should be made based on the response time specified in the system requirements and on response times calculated at the program design stage. These calculations will take into account the

processing and file accessing requirements of the messages in the system and the rate of arrival of these messages, so no generalization can be made.

A single threading program may later be converted to multi-threading without rewriting the processing element, providing the standards described in Chapter 3 are adhered to.

Standard routines

Alternative versions of standard routines are only available for a multithreading Driver. Details of the routines available may be found in Chapter 6. This section summarises the choices that must be made.

MESSAGE SCHEDULING

Multithreading Driver offers a choice of two message-scheduling systems within Bead Scheduler. TABs queued to be accessed by a bead may be serviced either on a *first in-first out* basis or according to a *bead priority* system. In the first system all TABs queued for beads are held in one queue, regardless of which bead they are waiting for, and are serviced in the order in which they arrive, i.e. the order in which they are held on the queue. In the "bead priority" system all tabs queued for bead 1 will be serviced before those queued for bead 2, etc. Alternative versions of Bead Scheduler are provided and one appropriate to the scheduling system selected should be consolidated into the program.

OVERLAY

Where insufficient main store is available to hold the entire real time program in main store, beads in the processing element may be overlaid from disc. When an overlay system is used, the programmer must incorporate into the Driver program the standard 1900 overlay package %EROL, and also,for a multithreading progam, the Overlay Control module described in Chapter 6. To enable the overlay of beads to be time shared with other processing during multithreading, Overlay Control functions autonomously in Member 2 of the program, and is activated, when required, by Request Analyser or Bead Scheduler. Alternative versions of Request Analyser and Bead

Scheduler are provided for overlaid and non-overlaid programs. (Four versions of Bead Scheduler are provided so that the choice of message-scheduling system is available in both overlaid and non-overlaid programs).

COMMUNICATIONS MONITOR

The communications handling routines in the Driver program may consist of the appropriate 1900 communications housekeeping package or user-coded routines. Communications Monitor provides the interface between these routines and the main program.

In a single threading program, Communications Monitor is always user-written. Details may be found in Chapter 5, page 29.

Standard multithreading Communications Monitor routines are provided to interface with the 7900 Message Buffering housekeeping package, the Mk II local Visual Display Unit housekeeping package and the Character Buffering housekeeping package. The interface routine(s) appropriate to the selected housekeeping package(s) will be supplied as required for inclusion in the user's program. If it is wished to use other housekeeping packages, or a user-coded communications handling routine in a multithreading program, the user must write his own interface routine for inclusion within Communications Monitor. Details of the multithreading Communications Monitor may be found in Chapter 6.

STORE ADMINISTRATION

Two methods of store allocation are available in 1900 Driver. In a *fixed store* system sufficient core for the processing of each possible message type is permanently allocated to each TAB in the system. Multithreading programs may alternativel use a *dynamic store* system in which the store necessary for

processing of a particular message is allocated during the processing of the message. A description of these two systems is given in the section *Alternative methods of store organisation* in this chapter.

If a dynamic store allocation system is required, the Store Administrator routine must be included in the program. Details of Store Administrator may be found in Chapter 6.

Beads written for a dynamic store multithreading program may also be used in a single threading program provided that a dummy Store Administrator routine is included, as described in Chapter 5.

USER-WRITTEN ROUTINES AND OWN CODING

Master routine

For any 1900 Driver program the user must code a Master routine. This routine,details of which are included in Chapter 4, performsthe following functions

1 It gives the program name, and the priority of Member 0.

2 It includes all compilation and consolidation directives required by the program.

3 It contains the data definitions of the TAB or TABS and associated areas, and all *standard locations* (see page 16) containing user-defined Driver constants, parameter areas, and tables.

4 It includes program entry points (the initial entry points, plus any entry points associated with error recovery procedures).

5 It sets up a TAB and makes the initial entry to Driver.

Monitor routines

In a single threading program, Peripheral Monitor and Communications Monitor are user-written as described in Chapter 5. Enhancement to multithreading should be considered when designing these routines; in the majority of cases a compromise will have to be found between ease of coding in the first instance and ease of enhancement for the changeover to multithreading. For example, by following the standards given in Chapter 6, it is possible to write Peripheral Monitor and its associated file handling routines in such a way that the latter routines can be run unchanged in a multithreading program. However, the extra effort needed to write the requisite dummy subroutines, set up dummy locations, etc. could delay implementation of the single threading program. In addition, the interpretation of request codes and parameters by the monitor routines before and after the changeover to multithreading should be carefully planned to avoid the need for major recoding during enhancement.

In a multithreading program, Peripheral Monitor is a standard routine which interfaces with user-written file handling routines as described in Chapter 6. These routines may in turn interface with Direct Access or Magnetic Tape housekeeping, or else be coded at PERI level.

User entry points

User entry points are provided at several points in the multithreading Driver routines to enable the user to incorporate his own coding in these routines, and to replace certain standard subroutines. In particular, Driver enters all queuing sub-routines by means of user entry points - this facilitates

replacement of these subroutines to cater for non-standard queuing requirements and scheduling. Entry points are also provided to allow user manipulation of certain parameters and reply information within Communications Monitor. Full details of these own coding options may be found in the description of the multithreading Driver in Chapter 6.

Replacement of standard routines

Where it is decided that certain standard routines do not meet special requirements, the user may code his own routines to replace them. For instance, the user may wish to write his own Bead Scheduler to implement a non-standard message-scheduling algorithm. Full details of the standard routines and interfaces between them may be found in Chapters 5 and 6.

Language

All control routines supplied by the user must be coded in PLAN.

THE PROCESSING ELEMENT

Choice of language

Beads may be written in PLAN or COBOL; PLAN and COBOL beads may be incorporated into the same program. The normal advantages of using COBOL all apply, and the language is extensively used in Driver programs. PLAN beads are coded as program segments; COBOL beads are written and compiled as subroutines, as described in Chapter 12 of the *1900 COBOL* manual.

Bead Size

It is not possible to give fixed or recommended sizes for beads. However, the following major factors should be considered when deciding how to divide the processing element into beads.

1 The program may split logically into independent units, (for example, vets, updates, enquiries) each of which can become a bead. This is the simplest method of structuring a Driver program.

2 A multithreading Driver will not allow a message to enter a bead until the previous message has made its final exit from that bead. Greater throughput may be achieved by terminating some or all beads at their first exit to Driver, thus freeing them for use by other messages. This reduces bead sizes, but increases the number of beads required.

3 Overlay requirements may have some bearing on bead size since the unit of overlay is always a bead. For example, all beads allocated to a given overlay area should be of approximately the same size if possible. Much depends here on accurate estimation of the size of beads.

4 Resource allocation may be a considereation. If few programmers are available, then writing a few large beads rather than a relatively large number of small ones may possibly result in greater programming efficiency.

System Beads

Whilst beads are normally specific to the processing required of the individual program, beads to perform certain control functions will be found in every 1900 Driver program.

Reference is made below to *bead numbers*: the means by which these numbers are assigned to beads is described in the section *Bead Branch Table*, Chapter 4, page 10 .

INITIAL BEAD

When a message has been received by a program and linked to a TAB by Communications Monitor, it will be passed to a nominated bead which will be the same bead for all messages. It is recommended that this practice is followed in a single threading program also. Bead 1 is conventionally the Initial bead in every thread, and its function is to recognise each message received and accordingly request entry to the first processing bead required in each case.

INITIALIZATION BEAD

Before the first message is received into the Driver program, certain initialization procedures must be performed, for example, opening files and allocating the communications terminals. Also, some form of introductory message will normally be output to inform terminal operators that they may now commence input. These procedures may be coded as part of the Master routine, though they will then be store resident. If it is wished to overlay them instead, they should be coded as a bead. The first entry to Driver from the Master routine will thus be a request to pass control to this Initialization bead.

ERROR RECOVERY BEAD

Errors detected by Driver control routines are processed by passing control to Bead 0. An error parameter will have been set in word 8 of the TAB as described in the descriptions of the individual routines in Chapters 5 and 6. Bead 0 should

therefore be coded as the first bead of the user's error processing thread(s).

COMMON SUBROUTINES

Subroutines called by several beads or file handling routines may be included in the permanent part of the Driver program as common subroutines. Such subroutines, however, must not access peripherals directly, nor must they issue any Driver request. These subroutines should be coded and compiled as a free-standing program segment or segments, or be included in the Master routine.

In a multithreading program there are several standard common subroutines accessed by the Driver control routines; details of these may be found in Chapter 6.

STORE ORGANISATION AND USE

TABs and associated areas

GENERAL DESCRIPTION

Whenever an input message is received into the program, Driver allocates to the message a 28-word area of store known as a Task Administration Block, or TAB. The TAB forms the interface between Driver and the beads required to process the message: during the message's life in the program, it holds all parameters generated by Driver and the beads for each other's use in processing the message. On completion of processing, the TAB is deallocated from its message by Driver, ready for allocation to another message.

A TAB is confined to holding the control information required by Driver and the beads during the processing of the message

to which the TAB is allocated. Further areas of store are required so that user data, including the original input message, can be accessed by the beads, and to enable reply messages and other data generated by the beads to be assembled and stored for output. Up to three such areas may be chained to the TAB by means of link addresses stored in three reserved words of the TAB. These areas are hence known as the *associated areas* of the TAB.

Each associated area of a TAB has a specific purpose, and is named accordingly, thus:

1 The Message area holds the input message to which the TAB is currently allocated, together with certain relevant parameters, for example, message length. During processing of an input message, each reply message generated by the beads is stored in this area, waiting to be accessed and output by Communications Monitor on receipt of the appropriate request from a bead. Note that throughout the processing sequence, the TAB is considered as being allocated to the input message, even though this message may be overwritten by a bead-generated reply message during processing.

2 The Input/Output area holds all data read in by Peripheral Monitor from backing store following a request from a bead for peripheral input. Similarly, any bead-generated data to be written to a file is stored in this area in preparation for output by Peripheral Monitor.

3 The Additional Core area is available, if required, for any other purpose. Typical examples of its use are as a work area for the beads, and as a means of storing data to be passed from one bead to another. There are two restrictions on its use:

(a) It must not be used for communications or peripheral input/output.

(b) It cannot be used to retain data once the TAB with which it is associated is deallocated. Data which is to be retained in the program for use in the processing of another message must be placed in one of the *User Common Areas* described on page 18.

TAB PARAMETERS

A TAB holds 4 main types of information:

1 Information provided by the beads for Driver. This information is provided each time a bead issues a request for a Driver Service: the following parameters are inserted by the bead into reserved locations of the TAB immediately before exit from the bead to Driver.

(a) A *request code*, defining the Driver facility required.

(b) *Request parameters*, to control the operation of the Driver routine providing the required facility.

(c) Optionally, a bead *re-entry parameter* (if the bead is to be re-entered following servicing of its current request).

(d) If the current request is for transfer of control to another bead, the bead number of the bead to be entered.

2 Information set up in the TAB by the Master routine for use by both Driver and the beads throughout the operation of the program. This consists of

(a) The start address of the TAB's Message Area.

(b) (Fixed store systems only - see page 19). The start addresses of the Input/Output and Additional Core areas permanently associated with the TAB.

3 Information placed in the TAB by Driver for use by the beads. This consists of

(a) The logical terminal number of the device from which the message currently being processed has been input.

(b) Contents of accumulators 0 and 4 to 7. The accumulators are stored in the TAB by Request Analyser on entry from a bead, and are restored immediately before control is passed to a bead.

(c) (Dynamic store systems only - see page 19). Following a store request from a bead, the start addresses of the *store cells* allocated to the TAB Store Administrator as Input/Output and Additional Core areas. The link to the Input/Output area is also used by Driver (Peripheral Monitor) when servicing peripheral requests.

(d) An *error reply parameter*. Any Driver routine detecting an error places a code defining the type of error in a reserved word of the TAB. This word is subsequently accessed by the Error Recovery bead (see page 9) in order to determine the recovery action required.

4 Information stored by Driver for its own use in controlling the processing of messages. This includes

(a) The bead number of the bead issuing the current request. Before control is passed to a bead, its bead number, as mentioned in 1(d) above, is shifted to an adjacent word of the TAB. Thus, when the bead which has been entered issues a request, Driver can determine its identity by examining the contents of this word.

(b) (Multithreading systems only). A *queue chain link address*. This is set up by the appropriate queuin subroutine within Driver whenever the TAB is queued awaiting a Driver service. It consists of the start address of the next TAB subsequently placed on the same queue.

(c) (Multithreading systems only). The permanent and temporary *processing priorities* of the message to which the TAB is currently allocated. These parameters are only used when the standard multithreading Driver routines are modified by the user to support a message priority system of servicing bead requests. Full details are given within the section *Queue manipulation*, Chapter 6.

(d) (Multithreading systems only). The start address of the *Queue Management Area* (QMA) for the Free TAB queue. Full information on the use of the Free TAB queue and QMAs in general is goven in the section *The queueing system* in Chapter 6.

TAB FORMAT

A TAB is a 28 word area which is set up by the user in his Master Routine, as described in Chapter 4. For a single threading program, one TAB is required. For a multithreadin

program, the number of TABs set up must be equal to the maximum number of messages that the program is to be capable of handling simulataneously.

During the operation of the program the various types of information described in the previous section are held in the TAB in the following format:

Word 0	Request code	Request area
Words 1 to 7	Request parameters	Request area
Word 8	Error reply parameter	Request area
Word 9	Logical terminal number	
Word 10	Bead re-entry point	
Word 11	Bead number of bead last entered/originating current request	
Word 12	Bead number of next bead to be entered	
Words 13 to 15	Spare	
Word 16	Contents of Accumulator 4	on last entry to/ exit from Driver
Word 17	Contents of Accumulator 5	on last entry to/ exit from Driver
Word 18	Contents of Accumulator 6	on last entry to/ exit from Driver
Word 19	Contents of Accumulator 7	on last entry to/ exit from Driver
Word 20	Contents of Accumulator 0	on last entry to/ exit from Driver
Word 21	Queue chain link address	
Word 22	Start address of Message area	
Word 23	Start address of Input/Output area	
Word 24	Start address of Additional Core area	
Word 25	Permanent priority of the message to which the TAB is currently allocated	
Word 26	Temporary priority of the message	
Word 27	Free TAB Queue Management Area address	

ASSOCIATED AREAS

The Message, Input/Output and Additional Core areas associated with a TAB are areas of any required length which are defined in the Master routine. A TAB must always have a Message area associated with it, the Message area being permanently chained to the TAB by means of a link address preset by the user in word 22 of the TAB. The Input/Output and Additional Core areas are optional, and the manner in which they are defined depends on whether a fixed or dynamic store allocation system is in use (see *Alternative methods of store organisation*, page 18).

The first 4 words of each area associated with a TAB are reserved for Driver control information. In practice, these four words are only used by the multithreading Driver when employing dynamic store allocation. The parameters used by Driver in this case are described in the section *Store cell queueing*, Chapter 6.

Beads written for simpler systems, however, will not be compatible with a dynamic store system if they access and store data in these words. Words 0 to 3 of each associated area should therefore be regarded as reserved for Driver use in all programs; this standard should also be observed by user control routines written for the single threading Driver.

STANDARD LOCATIONS

With the exception of the TAB(s) and associated areas, all store locations and areas used within Driver are defined under standard common blocknames beginning with the letters HDR.

Some of these locations must be preset in the Master Routine to enable Driver to perform the specific control functions required by the User's individual program. For example, entry to the beads by Driver is always carried out using a table of

CALL instructions; this table must be set up under the common blockname HDRBT1, and contain one CALL instruction for each bead present in the user's program.

Full instructions for defining and presetting HDR locations are given in Chapter 4. The three described below are of particular importance.

Address of current TAB (HDRTABSTORE)

For a single threading program, the location HDRTABSTORE is preset to hold the start address of the TAB. Both Driver and the beads access the TAB using this pointer.

In a multithreading program, all TABs are accessed on exactly the same principal. In this case, however, HDRTABSTORE is continually reset by Driver so that at any given moment it holds the address of the TAB whose message is currently being processed. The adjacent location HDRTABSTORE + 1 is reserved to hold the address of the current TAB if an error occurs during servicing: full details are given in the section *Exception Mode* in Chapter 6.

The Service TAB (HDRSCST)

The Service TAB is a 3 - word area with the common blockname HDRSCST. Parameters placed in this area by the Driver routines determine the sequence in which the routines are entered in the course of servicing a request, and also specify whether error recovery action will be necessary on return to the processing element.

In a single threading program, all user-written control routines provided as part of the Driver must be capable of updating the parameters in this area before exit. Use of the Service

TAB in single threading programs, and the settings required in exit from each routine, are fully described in Chapter 5.

Manipulation of the Service TAB parameters by the multithreading Driver is a somewhat complex operation. However, since the Driver modules which access the Service TAB cannot usefully be modified or replaced, the parameters have no practical significance so far as the user is concerned.

User Common Areas (HDRLUCDAT, HDRUUCDAT).

Data to be passed from one thread to another (that is, preserved after TAB deallocation) cannot be stored in any of the TAB's associated areas. If this facility is required, such data should be stored in a User Common Area starting at the common block HDRLUCDAT in Lower Storage or the common block HDRUUCDAT in upper storage.

ALTERNATIVE METHODS OF STORE ORGANISATION

Systems considerations

In a single threading program, the maximum size of each area of store required during processing governs the size of the associated areas to be defined for the TAB.

If there is sufficient store available for each TAB in a multithreading program to have areas of the maximum required size permanently chained to it, this is known as a fixed store system. Where messages have greatly differing store requirements, particularly where excessive store requirements occur for a small proportion of messages input, it may not be feasible to have a fixed store system in a multithreading environment and a dynamic store system is therefore employed.

Message area allocation

Message areas under both of the standard Driver systems of store allocation are all the same size and are each permanently allocated to a TAB. To meet the latter requirement, the user presets word 22 of each TAB to hold the start address of the Message area belonging to the TAB. Driver (Communications Monitor) always access a TAB's Message area via this pointer.

Fixed store system

This is the simplest system of store allocation. It involves the permanent allocation of sufficient store to give each TAB one Input/Output and one Additional Core area of the maximum size required at any time during the operation of the program. To meet these requirements, the user presets words 23 and 24 of the TAB (Input/Output and Additional Core area start addresses respectively) as for the Message area.

Dynamic store system

A dynamic store system is used when there is insufficient main store available to support a fixed store system. Under a dynamic system, Input/Output and Additional Core areas are allocated to a TAB by Driver (Store Administrator) on receipt of a store request from the bead currently accessing that TAB. The areas of store available for allocation in this way are set up by the user in his Master Routine, each area being termed a *store cell*. These cells collectively form the program's *common pool* of store.

The size and number of cells in a pool is of course dependent on the program's individual requirements. However, they must always be defined as belonging to one or more common blocks

of store, each block comprising a suitable number of equal size cells in a chained queue. Parameters enabling Driver to determine the location and current availability of the cells in each block must be set up in standard locations: full details are given in Chapter 4.

Whenever a bead accessing a TAB (i.e. processing a message) requires store, it issues a request for a cell of store from a particular block, hence implicitly specifying the size of cell required. Store Administrator allocates a cell from this block to the TAB by placing the start address of the cell in either word 23 of the TAB (Input/Output area requested) or word 24 (Additional Core area requested). The cell remains allocated to the TAB until a deallocation request is received by Driver, whereupon the cell is 'returned' to the pool by clearing its address from the TAB and manipulating the parameters controlling the organisation of the appropriate cell queue.

Since only two address locations are provided in the TAB, it is only possible to allocate one Input/Output and one Additional Core area to a TAB during the life of a message in the system. A bead may, if required, specify allocation of both these areas in a single request. If this is done, however, there is a risk of certain undesirable system conditions occurring: to keep this risk to a minimum, a standard method of defining store cells and sequencing bead-generated requests for store is recommended, and is described below. The actual coding required when defining store cells is described in Chapter 4.

AVOIDING STORE LOCKOUT

A condition known as store lockout occurs when allocation of a store cell to a TAB is delayed due to no cells of the required size being currently available, i.e. allocated to other TABs. Consequently the processing of the message to

which the TAB is allocated is also delayed. Store lockout is particularly likely to occur when two cells are required by a TAB: this is because only one cell can be allocated to a TAB at a time, a further delay being necessary before the second cell can be allocated. As a result, the first cell allocated may be locked out from the other TABs for an undesirably long period, due to the extra delay incurred whilst the TAB is awaiting allocation of the second cell.

The average time required for cell allocation can be kept to a minimum if the blocks of store cells are set up with the following facts in mind.

1 Under the standard Driver dynamic system, the various blocks of store cells are numbered in ascending order, starting at Block 0. If a bead written to Driver standards requests cells from two different blocks, Driver will obtain the first cell from the lower numbered block and the second cell from the higher numbered block. For example, if a bead requests cells from Blocks 0 and 4, the first cell will always be obtained from Block 0 and the second from Block 4.

2 As a general rule, the chance of a cell being allocated to a TAB within a given period depends on the number of cells in the block from which the cell is to be obtained. For example, if Block 0 contains one cell, and Block 4 contains six cells, there is a considerably better chance of a cell in Block 4 being available at any given time.

The user can therefore keep the delay between allocation of the first and second cells to a minimum by arranging the blocks according to the number of cells contained, in ascending order. For example a typical pool might be set up as shown below.

Example

Pool size 2012 words

Block 0:	1 x 256	word	cell
Block 1:	2 x 500	word	cells
Block 2:	4 x 64	word	cells
Block 3:	5 x 100	word	cells

AVOIDING BEAD LOCKOUT

If any bead were allowed to make requests for store cells, the possibility could arise of a bead making a request that could not be satisfied for a relatively long period, due to one or both of the required cells being allocated to other TABs. This condition is known as bead lockout, and will cause a considerable increase in the time taken to execute the bead, resulting in delays, to other messages requiring processing by that bead.

Therefore, either:

1 All store requests should be made by the Initial Bead. These requests should also specify transfer of control to another bead, thus freeing the Initial Bead to process another message.

2 Alternatively, if store requests are made from beads other than the Initial Bead, they should also specify transfer of control to another bead. This avoids the possibility of the bead issuing the request being locked out of the program.

The first method is particularly recommended since it avoids

the possibility of a bead attempting to access non-existent Input/Output and Additional Core areas. This could happen, if, due to an oversight in planning the message threads, a bead attempted to access these areas before they were allocated to the TAB.

AVOIDING TOTAL PARALYSIS

It is a Driver standard that beads requiring allocation of two cells (i.e. both an Input/Output and Additional Core area) to a TAB must request these cells in ascending sequence of block number. If this is not done, i.e. cells are requested in random block sequence, it is possible that mutual store lockout will occur between TABs, resulting in total paralysis of the program.

Example

Referring to the example on page 22, suppose that:

1 4 TABs have had store allocated from block 2 and each of these TABs is now awaiting allocation of a second cell from block 1. The cells have thus been requested in descending block sequence.

2 2 TABs have had store allocated to them from block 1 and are each awaiting allocation of a second cell from block 2.

It can seem that, since all the cells in both blocks are allocated, the second cell requirements of the two sets of TABs are mutually exclusive. Processing of the messages to which these TABs are allocated will therefore not continue.

IMPLEMENTATION

This section contains information specific to 1900 Driver which should be used in conjunction with the *Programming Procedures* manual when preparing specifications, estimates and a project timetable for the Driver program.

Specification of user-written routines

All user-written control routines are subroutines of the program, so specification of these should follow the standards for subroutine specification as described in *Programming Procedures*. Beads are also called as subroutines, so the subroutine specification standards apply; however, as they perform functions similar to programs in a batch suite, some portions of the program specification standards may also be applicable, depending on the design of the individual user's program.

ESTIMATING PROGRAM SIZE

Program composition

A 1900 Driver program will include the following:

Master routine, plus any initialization routines

Driver control routines (Request Analyser, Bead Scheduler, System Control)

Tables and constants required by Driver (for example, bead tables)

Peripheral Monitor (supplied as standard control routine in multithreading Driver)

File Handling Routines (if used)

Peripheral control routines, including any control areas and buffers

Communications Monitor (supplied as standard control routine in multithreading Driver)

Communications control routines, including any buffers, etc.

Overlay Control (if used - multithreading programs only)

Overlay routines (if used)

Store Administrator (if used)

TABs and associated areas, comprising

- 28 word TABs
- Message Areas
- Input-Output Areas
- Additional Core Areas

} one of each per TAB

Dynamic store common pool (if used)

User common Areas

Beads (some may be overlaid)

Driver common subroutines (fultithreading only)

User common subroutines (if used)

Sizes of standard Driver control routines

The three standard control routines for single threading Driver occupy less than 100 words of store. A multithreading Driver, using standard routines throughout, will occupy up to 2000 words of store. Detailed information on th sizes of individual routines may be found in Appendix 3.

Housekeeping routines

Details of communications housekeeping packages are given in the manual *Data Communications and Interrogation*. Details of peripheral housekeeping routines are given in the manuals *Direct Access* and *Magnetic Tape*. Buffer sizes must be included when calculating store requirements.

Total program size

The store requirements for a non-overlaid Driver program will be the sum of the sizes of the appropriate areas listed under *Program Composition* above. For an overlaid program allowance must be made for some beads being overlay units; the other areas listed will be in the permanent part of the program.

ESTIMATING EFFORT

Beads

When using Driver, a bead corresponds to a program in a batch suite. Thus the complexity factor for a bead, for estimating purposes, should be the same as for a batch program performing the same function. When coding a bead the programmer must conform to Driver standards, but has no file handling or terminal handling routines to code.

Control routines

The Peripheral and Communications Monitor routines for a single threading program and the file handling routines for a multithreading program are more complex than beads, so a complexity factor greater than 1 should be used for estimating purposes. The effort required will depend on the experience of the programmers involved, for instance, Communications housekeeping package, Peripheral Monitor requires experience of file handling.

It is recommended that the most expereienced PLAN programmer be given the task of coding these control routines, other programmers coding the beads.

Writing the Master routine for a single threading program is a very small task once the program is specified. One or two days of effort should be sufficient. Figures for a multi-threading Master routine are not yet available.

Testing

The majority of the testing of a 1900 Driver program will be carried out using the Driver Testing Aids. An extra few days effort should be allowed to enable the programmers involved to familiarise themselves with the Testing Aids. It is also suggested that the first bead be tested as early as possible to allow at least one programmer to gain experience of setting up and using the testing system.

Once programmers are able to use the Testing Aids, testing should proceed much the same as any other testing system. Beads are normally tested individually, then link-tested. The last stage of testing will be on-line testing; this is more complex than batch program testing, but it is difficult to generalize about the effort required. The effort will depend on the complexity of the terminal network; for instance, a small network of local terminals presents less problems than a large remote network.

Chapter 3

Beads

GENERAL

Beads may be written as either PLAN segments or COBOL subroutines. Since, with certain exceptions, all functions of the user's program other than the processing of data are delegated to the Driver, a bead normally consists entirely of application coding, interspersed with requests for Driver Services at those points in the routine where input/output, additional workspace, or transfer of control to another bead is required.

Beads issue requests for Driver services using a parameter area (the request area of the TAB) provided for this purpose. Each bead performs its allotted function(s) within the program by carrying out a logically complete set of operations on data submitted to it in the user defined areas associated with the TAB, namely the Message, Input/Output and Additional Core areas.

As mentioned in the previous chapter, TABs and their associated areas are the only means by which individual beads can communicate with their environment. This fact, together with the restrictions on bead functions outlined above, mean that certain coding standards must be observed if a bead is to operate successfully in a Driver based program. Otherwise, the only difference between a bead and the equivalent batch processing routine is that a bead must initiate all non-processing functions required during its execution by means of requests passed to Driver. The remainder of this chapter describes the mandatory coding standards and the procedure for issuing requests. Beads written to these standards may be run without modification

under both the single threading and multithreading versions of Driver.

RESTRICTIONS ON BEAD FUNCTIONS

Beads must not contain file definitions.

As a general rule, they may not execute any instructions which cause an Executive interrupt; for example, input/output transfers. For this reason, PLAN instructions with function codes in the range 150 to 161 inclusive may not be used. The one exception to this rule is the Initialization bead, which, if used, may carry out file and communications handling in the conventional manner when initializing the system at the beginning of the run. This bead is described on page 16.

Beads may not contain any of the following directives:

#ENTRY

#PMODE

#PERIPHERAL

#OVERLAY

#PERMANENT

#ERRORSEG

#ELASTIC

SUBROUTINES

Provided that it conforms to the standards given in the previous section, a subroutine may be held as part of the bead requiring it.

USE OF ACCUMULATORS

Driver uses all the accumulators, but accumulators 0 and 4 to 7 are preserved on entry to Driver and are restored on exit to the next bead. These accumulators may hence be used to pass data from one bead to another. Preservation of accumulators 1 to 3 by PLAN beads is the responsibility of the user, and may be accomplished by storing the contents of these accumulators in the Additional Core area before issuing a request.

ACCESSING A TAB AND ITS ASSOCIATED AREAS

Introduction

A bead which contributes to the processing of a particular message shares the following COMMON areas with the other beads processing the message.

1 The TAB allocated by Driver to the message.

2 The Message, Input/Output, and Additional Core areas associated with that TAB.

In a single threading system, where there is only one TAB, PLAN beads could address any of these areas directly, using the symbolic names assigned to the areas in the Master routine (see Chapter 4, page 4). However this method cannot be used by COBOL beads, nor is it suitable for use in multithreading systems, where a bead may have to access any one of several different TABS and their associated areas, depending on which message is currently being processed by the bead. Moreover, in systems employing dynamic store allocation the location of the Input/Output and Additional Core areas during the processing of a particular message depends on which cells of store are available in the common

pool, and hence cannot be predicated with any certainty.

The method of indirect addressing described below enables a bead to access any TAB and its associated areas regardless of circumstances. Single threading users writing PLAN beads are strongly recommended to use this method, since it enables PLAN beads to be run without modification under both the single and multithreading versions of Driver.

Method

COBOL BEADS

For COBOL beads the following addresses are supplied by Driver on entry, in the order given.

Start address of TAB

Start address of the Message Area

Start address of the Input/Output Area

Start address of the Additional Core Area

Start address of Lower User Common Area (HDRLUCDAT)

Start address of Upper User Common Area (HDRUUCDAT)

To enable a COBOL bead to access these areas, they must be defined, in the order shown, in the bead's Linkage Section. Thus the first 01 level definition should refer to the TAB (this may be followed by 01 level defintions REDEFINING the TAB) and should be followed by 01 level definitions for the other five areas.

Alternatively the areas may be defined in the Linkage Section in any sequence, in which case they must also be declared, in the order indicated above, in the USING clause of the Procedure Division. This is the standard COBOL subroutine

interface, as described in Chapter 12 of the *1900 COBOL* manual.

PLAN BEADS

A PLAN bead may address a TAB and its associated areas by either of the following methods

1 At any point during the execution of a PLAN bead, the start addresses of the TAB and associated areas currently to be addressed by the bead can be found in the following locations

	Location
Start address of TAB	HDRTABSTORE
Start address of Message area	Word 22 of TAB
Start address of Input/Output area	Word 23 of TAB
Start address of Additional Core area	Word 24 of TAB

Note that:-

(a) In single threading programs, the addresses in HDRTABSTORE and words 22 to 24 of the TAB are preset in the Master Routine

(b) In all multithreading programs, HDRTABSTORE is set by Driver each time control is about to be passed to a bead, hence specifying which one of the available TABs is to be accessed by the bead during its execution.

(c) In multithreading programs employing a fixed store system, the addresses in words 22 to 24 of each TAB are preset in the Master routine, as for a single threading program.

(d) In multithreading programs employing a dynamic store system, words 23 and 24 are not preset, but are set by Driver following a store request from a bead. When this has been done, the two words respectively contain the start addresses of the store cells allocated to the TAB by Driver as Input/Output and Additional Core areas.

A bead using the above method of addressing may hence be run without modification under both single and multithreading Driver systems.

2 A PLAN bead may alternatively obtain the address of the current TAB and associated areas using the parameters supplied by Driver for COBOL beads. On entry to a bead, the instructions

```
OBEY    0(1)
OBEY    1(1)
OBEY    2(1)
OBEY    3(1)
```

will respectively place the start address of the TAB, Message, Input/Output or Additional Core area into accumulator 3. This method is also compatible between single threading and multithreading systems.

PROGRAMMING PROCEDURE

Entry and re-entry to a bead

A bead is always entered at the first instruction. If a bead is to issue more than one request during its execution, it must set word 10 of the TAB before each exit to Driver, the parameter placed in this word being used by the bead on re-entry to determine the point within itself at which processing is to continue.

Whenever control is passed to a bead from another bead, word 10 of the TAB is set to zero, indicating that the bead issuing the request has completed its execution and is free to access another TAB. All beads requiring re-entry must conform to the following standards:

1 The value in word 10 of the TAB must be incremented by 1 on each exit from the bead which is to be followed by re-entry. Thus on the first such exit word 10 must be set to 1, on the second exit it must be set to 2, and so on.

2 For the final exit from the bead, word 10 of the TAB must be reset to zero.

3 The first action taken by the bead each time it is entered must be to use the value in word 10 of the TAB to branch to the appropriate point in the application coding. Since the interpretation of this parameter on entry is self-evidently a bead function, the precise means by which this is done is at the discretion of the user. Possible methods of using the value in word 10 include

   (a) As a modifier for an OBEY instruction on a table of branch instructions to the various entry points

within the bead (PLAN)

(b) As an identifier in a GO TO .....DEPENDING ON statement (COBOL)

Bead tidying

Care should be taken to ensure that all variables within a bead which influence the bead's execution are cleared or reset to their original values between finishing the processing of one message and proceeding with the processing of the next message. Where necessary, the sequence of instructions which is entered following initial entry to the bead (i.e. word 10 of the TAB = 0) should be coded so that any pointers, count values, modifiers etc. which may have been updated during processing of a previous message are reset before any further action is taken.

Issuing requests

A bead issues a request by setting up parameters in reserved locations of the TAB and passing control to Driver.

PARAMETERS

To issue a request, a bead must access the TAB (as described on page 3) and place the following parameters in the locations indicated.

		TAB locations
1	Request code	Word 0
2	Request parameters (if any)	Words 1 to 7
3	Bead sequencing parameters	Word 10 or Words 10 and 12

A list of recommended request codes and parameters is given in Appendix 2 .

Request code

A request code consists of 4 characters

faaa

where

f is a *facility code*

aaa is an *additive code*

The facility code character is normally in the range 1 to 3 It specifies the Driver service required and hence, implicitly, the control routine within Driver which is to be entered in order to initiate servicing of the request. The alternative settings are as follows

Facility Code = 0 Bead (Scheduler) request

= 1 Peripheral (monitor) request

= 2 Communications (monitor) request

= 3 Dynamic store allocation (Store Administrator) request.

Note:- Further facility codes may be used if additional user-written control routines are present within the Driver, provided that the appropriate modifications have been made to the System Control routine (see *System control requirements*. Chapter 4, page 10)

The additive code gives additional details, where necessary, of the Driver service required. The significance of each character of the code depends on the type of request being issued, and is therefore described for each request type in the appropriate sections of this Chapter (page 11 onwards)

Request parameters

The request parameters are supplementary to the request code and provide all the additional information required by the control routine which will service the current request. The parameters required for each type of request are described in pages 11 to 16.

Bead sequencing parameters

If a bead is to be re-entered following servicing of the current request, word 10 must be set to indicate the bead re-entry point, as described in the section *Entry and re-entry to a bead* on page 7 .

If control is to be passed to another bead, words 10 and 12 of the TAB are set as follows:

Word 10	0
Word 12	Bead number of the bead to which control is to be passed.

Note:- Word 11 of the TAB is reserved for use by Driver and must not be corrupted.

EXIT TO DRIVER

PLAN beads

A PLAN bead always passes control to Driver by branching to the Request Analyser routine. This is done by means of the instruction

```
BRN HDRRACUE
```

COBOL beads

Exit to Driver from COBOL beads is always to bead Scheduler and occurs on the statement.

EXIT PROGRAM

Bead Scheduler then branches to Request Analyser in the same manner as a PLAN bead.

BEAD REQUESTS

There are only two Driver functions which need be requested in this category

1 Pass control to another bead (request code 0000)

2 Free a bead to access another TAB (request code 0001)

The first of these is self-explanatory: no request parameters are required. The second is only used in multithreading systems and is only issued by the Error Recovery bead. This latter request may be required if a bead has issued a request and a serious error has occured whilst Driver is servicing it. If this happens, control will be passed to the Error Recovery bead, which may issue a request for a bead to be freed from its current TAB so that it is free to access another TAB and hence process another message. In this case, the Error Recovery bead must place the bead number of the bead concerned in word 1 of the TAB before passing control to Driver.

PERIPHERAL REQUESTS

Most of the parameters placed in the TAB by a bead issuing a peripheral request are for use by file handling routines, which are always user written. Accordingly, the nature and format of these parameters, and their location in the Request

area of the TAB, is largely decided by the user. However, the standards given below must be observed.

Request code

		Significance
Character 0		Facility code (always 1)
Character 1	Bits 9, 10 and 11 of the request code when set to 1, are of special significance as optional additive codes, as follows:	
	bit 9 = 1	More than one file is to be accessed during servicing of the request.
	bit 10 = 1	The TAB is to be deallocated after the request has been serviced.
	bit 11 = 1	Return to normal mode is to be effected after the current request has been serviced (multithreading programs only - see Chapter 6, page)
Character 2 and 3		Available for user parameters: otherwis zero.

Request parameters

Word 1 of the TAB must contain a parameter specifying the file to be accessed, typically a *file reference number* (see Chapter 5, page 26)

Word 3 of the TAB must contain a displacement value which when added to the start address of the Input/Output area (in word 23 of the TAB) gives the address within the Input/Output area where input or output of data is to begin.

COMMUNICATIONS REQUESTS

The main purpose of communications request codes and parameters is to provide the parameters required by Communications Monitor in issuing housekeeping macros. A full list of the codes and parameters for the package interface routines at present available within the multithreading Communications Monitor is given in Appendix 1. For requests issued to a single threading Communications Monitor routine, or user-designed interface routines, the following standards must be observed.

Request code		*Significance*
Character 0		Facility code (always 2)
Character 1		
	Bits 6 to 8	Package identification number
	Bits 9, 10 and 11 of the request code, when set to 1	

		Significance
	are of special significance as additive codes, as follows:	
	Bit 9 = 1	A message is to be output to, or some other action performed in respect of, more than one device.
	Bit 10 = 1	The TAB is to be deallocated after the current request has been serviced
	Bit 11 = 1	Return to normal mode is to be effected after the current request has been serviced (multithreading programs only - see Chapter 6, page)
Character 2		Reserved for a function identifier, specifying the type of activity required, e.g. input or output
Character 3		Available for further parameters defining the action required in more detail: otherwise zero.

Request parameters

If output to only one terminal is required, word 1 of the TAB must be set to contain the appropriate device identifier. Otherwise, this word must be set to zero.

Where relevant, word 2 of the TAB must hold the length in characters of the message to be output.

Where relevant, word 3 must contain a displacement value which when added to the start address of the Message area (held in word 22 of the TAB) will give the address of the first word of the message to be output.

If a message is to be output to, or some other action performed on, more than one terminal, word 7 of the TAB must contain the start address of the user's *device table* (see Chapter 4, page) and word 6 set as a counter/modifier to progress down this table.

STORE REQUESTS

Request code

There are only two requests in this category

1 Allocate store (request code 3010)

2 Deallocated store (request code 3020)

Request parameters

Each request may relate to one or two cells of store, these cells constituting an Input/Output and/or an Additional Core area.

Two parameters, occupying two words of the TAB, are required for each cell. The parameters are stored starting at word 1 of the TAB. Each pair of parameters comprises the block number of the block containing a cell of the required size, followed by the *destination address* (23 or 24) giving the word of the TAB in which the start address of the cell is to be placed (on allocation) or found (on deallocation). Giving the destination address 23 will hence cause the selected cell to be allocated as an Input/Output area: the destination address 24 will similarly cause the cell to be allocated as an Additional Core area.

Where two cells are to be allocated or deallocated following a single request (i.e. two pairs of parameters are stored in the TAB) words 1 and 2 of the TAB must contain the parameters for the first cell to be allocated, or deallocated, whilst those for the second cell must be held in words 3 and 4. The blocks in which these cells are held must be referenced in ascending order of block number, for the reasons given in the section *Dynamic store system*, Chapter 2, page 19.

SYSTEM BEADS

System beads are so called because they play a significant part in the overall operation of the user's system instead of being confined to processing data. As explained in Chapter 2, the majority of Driver programs contain three system beads: an Initialization bead, an Initial bead, and an Error Recovery bead.

Initialization bead

Use of this bead enables initialization procedures to be overlaid. The bead carries out any initialization procedures

not contained within the Master routine.

The Initialization bead differs from conventional beads in that it may, if required, directly execute PERI instructions and issue housekeeping macros, thus avoiding the need to code initialization functions, such as opening files and activating terminals, within the Master routine or the control routines. Its final action, however, must be to issue a communications or peripheral request specifying deallocation of the TAB: if this is not done, input from the terminals will not begin.

Initial bead (bead 1)

Driver does not identify the processing requirements of messages input to the program: this is hence a bead function. The user is recommended to write a bead, known as the Initial bead, which will always be the first bead entered whenever processing of a message beings. The function of the Initial bead is to identify the thread required to process each input message, and transfer control to the first bead in the appropriate thread.

Error Recovery bead (bead 0)

If any of the control routines detect errors which prevent them from continuing the servicing of a request, they will pass control, either directly or use System Control and Bead Scheduler, to bead 0. The routine detecting the error will also place an error code in word 8 of the TAB.

The Error Recovery bead must be written so that it will examine the contents of word 8 of the TAB and accordingly take or attempt the appropriate error recovery action.

EXAMPLES

The PLAN and COBOL beads shown below will each set up a start of day message starting at word of the Message area, request output of this message to terminals 1, 2, 3, 8 and 9, and also request deallocation of the TAB. Note that:

1 In both beads, character 1 of the request code (>) sets bits 6 to 11 of the code to the pattern 001110 (#34). The significance of these settings is as follows:

Bits 6 to 8 (=001)	7900 package identification number (=1)
Bit 9 (=1)	Multiple output required
Bit 10 (=1)	Deallocation of TAB required
Bit 11 (=0)	Return to normal mode option not required in this case

2 The COBOL bead sets up the terminal list in word 5 onwards of the Additional Core area, thus enabling it to set up the address of this list by accessing the Additional Core area start address stored in the TAB

PLAN

```
#PROGRAM        ZZZZ50/BEAD07
#LOWER          COMMON/HDRTABSTORE/
                TABAD
#LOWER
MESS            28HSYSTEMVREADYV-VGOODVMORNING!
LIST            1,2,3,8,9,-1            [TERMINAL LIST
RQST            4H2>10                  [REQUEST CODE
                0,28,5,#100,1,1,0/LIST
#PROGRAM
      LDX   2   TABAD                   [PICK UP TAB ADDRESS
      LDN   1   RQST
      MOVE  1   8                       [MOVE REQUEST TO WORDS 0-7 OF TAB
      LDX   1   22(2)                   [PICK UP MESSAGE AREA ADDRESS
      ADN   1   5
      LDN   0   MESS
      MOVE  0   7                       [MOVE MESSAGE TO MESSAGE AREA
      ST0Z      10(2)                   [ZEROISE RE-ENTRY POINT
      BRN       HDRRACUE                [EXIT TO DRIVER
#END
```

COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BEAD07.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
    SOURCE-COMPUTER. ICL-1904.
    OBJECT-COMPUTER. ICL-1904.
    MEMORY SIZE 13000 WORDS.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  BROADCAST.
  02  MESSAGE PIC X(28) VALUE "SYSTEM READY - GOOD MORNING!".
  02  REQUEST
    03  WD-0  PIC X(4)  VALUE "2>10".
    03  WD-1  PIC 1(24) VALUE ZERO.
    03  WD-2  PIC 1(24) VALUE 28.
    03  WD-3  PIC 1(24) VALUE 5.
    03  WD-4  PIC 1(24) VALUE 64.
    03  WD-5  PIC 1(24) VALUE 1.
    03  WD-6  PIC 1(24) VALUE 1.
  02  LIST
```

COBOL

```
        03  TM-1   PIC 1(24) VALUE 1.
        03  TM-2   PIC 1(24) VALUE 2.
        03  TM-3   PIC 1(24) VALUE 3.
        03  TM-8   PIC 1(24) VALUE 8.
        03  TM-9   PIC 1(24) VALUE 9.
        03  TM-Z   PIC S1(23) VALUE -1.
LINKAGE SECTION
01  TAB
    02  REQ-AREA      PIC X(28).
    02  WD-7          PIC 1(24).
    02  FILLER        PIC X(8).
    02  WD-10         PIC 1(24).
    02  FILLER        PIC X(52).
    02  WD-24         PIC 1(24).
    02  FILLER        PIC X(12).
01  MESS-AREA.
    02  FILLER        PIC X(20).
    02  DATA          PIC X(28).
01  IO-AREA.
    02  FILLER        PIC X.
01  ADD-CORE
    02  FILLER        PIC X(16).
```

COBOL

```
    02  LIST-2       PIC X(24).
01  UCA-1.
    02  FILLER PIC X.
01  UCA-2.
    02  FILLER PIC X
PROCEDURE DIVISION.
L1.
    MOVE REQUEST TO REQ-AREA.
    MOVE WD-24 TO WD-7.
    ADD 4 TO WD-7.
    MOVE MESSAGE TO DATA.
    MOVE LIST TO LIST-2.
    MOVE ZERO TO WD-10.
    EXIT PROGRAM.
```

****

Chapter 4

The Master routine

INTRODUCTION

The Master routine is written by the user in PLAN and is resident in main store during the running of the program. It carries out the application orientated tasks required by the user's system before entry is made to the main program.

The Master routine must contain the following information.

1 Program name/priority

2 All compilation and consolidation directives required by the program as a whole (e.g. #PERMANENT, #LIBRARY)

3 Definitions of all data areas to be preset before entry to the program (for example, the TAB(s) and associated areas)

4 Program entry point(s) and, optionally, program end point.

It may also:

5 Allocate peripherals and open all files to be used by the program.

6 Carry out the initial activation of any other program members to be used.

7 Activate the communications network

8 Carry out any additional initialization procedures required by the user, for example broadcasting start-of-run messages to terminals

In some systems, incorporation of last four types of function into the Master routine may result in the routine occupying an unacceptably large area of main store during the running of the program. Such functions may instead by overlaid using an Initialization bead, which is brought into main store and executed once the program has been entered from the Master routine. This bead is described in Chapter 3, page 16.

PROGRAMMING PROCEDURE - SINGLE THREADING PROGRAMS

The standards for writing a Master routine for a single threading program are described below. The standards are similar for a multithreading program, the main difference being that certain types of area are defined more than once. For example, a different TAB will have to be defined for each message type to be processed. The additional area definitions and other coding required for multithreading are described from page 17 onwards.

System organisation

COMPILER DIRECTIVES

All compiler directives required by the program must appear in the Master routine. These will include:

1 A #PERMANENT directive for the 1900 Driver routines in the program. The single threading 1900 Driver routines are listed under this directive using the following names.

Routine	*Name*
Request Analyser	HDRRA1
System Control	HDRSC1
Bead Scheduler	HDRBS1

2 All communications housekeeping subroutines required by the program must similarly be listed under a #PERMANENT directive.

3 If a PLAN compiler with magnetic tape output is to be used to compile and consolidate the program, the library subfile SDRV holding the 1900 Driver control routines must be listed under a #LIBRARY directive. All other library subfiles containing routines to be used by the program, including those containing subroutines in the S-RS group, must also be specified under this directive.

4 For an overlay program, the Master routine must be preceded by the *Overlay steering segment*, which must be the first segment presented to the consolidator. This segment will comprise the following items:

(a) The program name and priority of program number 0, under a #PROGRAM directive.

(b) A #PERMANENT directive for the overlay package %EROL

(c) A #OVERLAY directive followed by a list of the overlay beads and their area/unit numbers in the format described in the *PLAN reference manual* Edition 1, T.P. 4004 Chapter 6, page 23.

Note that the name and priority of the complete program will be as defined in (a) above.

ACTIVATION OF OTHER PROGRAM MEMBERS

If members other than member 0 are to be used by the program, their initial activation should be carried out from within the Master routine. This rule does not apply to the member

containing the Communications housekeeping: the macros required to establish and implement the housekeeping member may alternatively be coded either within the Initialization bead or within an initialization subroutine which is executed on the first entry to Communications Monitor.

Main store organisation

TAB AREA DEFINITIONS

The TAB and its associated areas may be defined using any desired symbolic names, provided that names beginning with the letters HDR are not used.

Mandatory areas:-

1 x TAB (length 28 words)
1 x Message area (of maximum length required)

Optional areas

1 x Input/Output area (of maximum length required)
1 x Additional Core area (of maximum length required)

TAB presets

When defining the TAB, the request area and Words 10 and 12 of the TAB may be preset for initial entry to the program as described in the section *Activating the Driver*, on page 16.

Both PLAN beads and the Driver control routines will use link addresses held in the TAB to locate the TAB's associated areas. These link addresses are set up by storing the start addresses of the various areas in the following locations

	Location
Start address of Message area	Word 22 of the TAB
Start address of Input/Output area	Word 23 of the TAB
Start address of Additional core area	Word 24 of the TAB

Message area presets

Words 0 to 3 of the Message area are reserved for Driver use: Word 0 must be preset to contain the total length of the area in words and words 1 and 2 set zero.

Following input of a message, word 4 of the area will be set by Driver to a value equal to the total length if the message in characters. It is therefore suggested that this word be initially set zero and not used for any other purpose. Any preset parameters which may be required by the beads when accessing the Message area should be stored in contiguous words of the area starting at word 5.

If a start-of-run message is to be output on activation of the Driver, the message should be specified within the Message area definition.

Input/Output and Additional core area presets

When defining either the Input/Output or Additional Core area, word 0 must be preset to give the length of the area in words. Words 1 to 3 are reserved for Driver-generated control parameters and must be preset to zero.

HDRTABSTORE

The start address of the TAB must be set up in the common block HDRTABSTORE. This word will be examined by both the control

routines and PLAN beads in order to locate the TAB in main store.

Example

```
#UPPER

10          516,0,0,0           [INPUT/OUTPUT AREA
            PERIDATA(512)

AC          49,0,0,0            [ADDITIONAL CORE AREA
            WORKSPACE (45)

MESS        128,0,0,0,0         [MESSAGE AREA
            MESSAGE (124)


            TAB (22)            [TAB

            O/MESS,O/IO,O/AC    [ADDRESSES IN WORDS 22-24 OF TAB
```

```
                SPAREWORDS (3)          [BRINGS TAB UP TO REQUIRED LENGTH (28 WORDS)

#LOWER          COMMON/HDRTABSTORE/

TABAD           0(TAB                   [ADDRESS OF TAB IN HDRTABSTORE
```

User Common Areas (HDRLUCDAT, HDRLUUDAT)

The User Common Areas HDRLUCDAT and HDRUUCDAT are declared as Common blocks of any required length under a #LOWER and #UPPER directive respectively. If either of these areas is not required, it must still be declared as a one word area to avoid missing cues in the program when it is consolidated.

Other user-defined locations and areas

A number of locations and areas used by individual control routines must be preset before entry to the main program. They must be defined as Common blocks under a #LOWER directive.

REQUEST ANALYSER REQUIREMENTS

Facility code constant HDRRACONST

A location HDRRACONST must be provided for use by Request Analyser in vetting requests (see Chapter 5, page 6). Within the Master routine, HDRRACONST must be set to a value one greater than the highest facility code used in the program (see Chapter 3, page 9). In the majority of cases, HDRRACONST will be set to 3 (4 if a dummy Store Administrator routine is present in the program).

SYSTEM CONTROL REQUIREMENTS

System Control does not use any user defined locations unless the standard 1900 Driver and user written control routines are augmented by further routines in order to meet special system requirements. In this case, one additional branch instruction must be appended to the System Control Second Branch Table HDRSCBR2 for each additional routine, the format of the instruction being

BRN *cuename*

where *cuename* is the entry point/cue name of the routine. The branch instructions are declared in the Master routine starting at the location HDRSCBR2+4. Use of this table by System Control is described in Chapter 5, page 11.

BEAD SCHEDULER REQUIREMENTS

Highest bead number constant (HDRBDCONST)

A one word area HDRBDCONST is used by Bead Scheduler to check that a valid bead number has been specified in a request for transfer of control to a bead. HDRBDCONST must be set to a value one greater than the highest bead number in the program.

Bead branch table (HDRBTI)

The bead branch table HDRBT1 consists of a one-word entry for each bead in the program. If required, it may also contain dummy entries for beads that are to be incorporated into the program at some future date.

An entry for a bead already present in the program consists of a CALL instruction of the form

CALL 1 *beadname*

where *beadname* is the name of the bead.

An entry for a bead not present in the program may be any value or string of characters that can be stored in one word, but is conventionally given as zero.

When servicing a request for transfer of control to a bead, Bead Scheduler will branch to the appropriate CALL instruction in the table using the bead number specified by the user as a modifier (see *Bead Scheduler*, Chapter 5, page 15). Beads must therefore be referenced by the CALL instructions in ascending order, according to the bead numbers allotted to them by the user. Thus the first CALL in the table will always be to the Error Recovery bead (Bead 0).

Example

```
#LOWER          COMMON/HDRBT1/

BT1   CALL  1   ERRORCOPE          [BEAD 0 - ERROR RECOVERY BEAD
      CALL  1   ALLMESSAGES        [BEAD 1 - INITIAL BEAD
      CALL  1   INVOICEPRINT       [BEAD 2
                0,0,0              [BEADS 3 TO 5 - DUMMY ENTRIES
      CALL  1   RECEIPTVET         [BEAD 6
      CALL  1   INVOICEVET         [BEAD 7
      CALL  1   CREDITCHECK        [BEAD 8
      CALL  1   RECEIPTCHECK       [BEAD 9
      CALL  1   INVOICECHECK       [BEAD 10
                0,0,0,0            [BEADS 11 TO 14 - DUMMY ENTRIES
```

```
      CALL 1      RECEIPTPRINT          [BEAD 15

      CALL 1      INITIALIZE            [BEAD 16

#LOWER            COMMON/HDRBDCONST/    [HIGHEST BEAD NUMBER CONSTANT (=16+1)

BDCON             17                    [ =17
```

Bead Storage Table HDBRT2

The bead storage table contains a one word entry for each of the beads referenced in the Bead Branch Table, including those referenced by dummy entries. As with the Bead Branch Table, entries must be made in ascending order of bead number, starting with the entry for bead 0.

The entries in the Bead Storage Table classify the beads as follows:-

Store resident
Overlay
Non-existent (i.e. not yet present in the program)

Before passing control to a bead (by means of the appropriate CALL instruction in the Bead Branch table), Bead Scheduler will access the corresponding entry in the Bead Storage Table to check if the bead is available in main store, taking suitable action if it is not (see Chapter 5, page 16).

Each entry in the table is set up as follows

Bead type		*Entry (1 word)*
1	Store resident	0
2	Overlay	area/unit number
3	Non-existent	immaterial

EXAMPLE

The following coding will set up a bead storage table for the beads referenced in the previous example (page 12). For the purposes of this example, ERRORCOPE, ALLMESSAGES, CREDITCHECK and RECEIPTPRINT are defined as being permanently resident in main store, and all other beads currently present in the system are defined as overlays.

```
#LOWER          COMMON/HDRBT2/

BT2             0,0,3/1             [BEADS 0 TO 2

                DUMMY1 (3)          [BEADS 3 TO 5

                2/2,2/1,0,1/2,1/1   [BEADS 6 TO 10

                DUMMY2(4)           [BEADS 11 TO 14


                0,1/3               [BEADS 15 AND 16
```

PERIPHERAL MONITOR REQUIREMENTS (OPTIONAL)

If the user's Peripheral Monitor routine is written so that file handling functions are delegated to separate routines, it may make use of a table of branch instructions in order to enter the required file handling routine(s) following each peripheral request. The user can set up this table either within the Master routine or within Peripheral Monitor itself. Further details and an example are given in the description of Peripheral Monitor in Chapter 5, page 26.

COMMUNICATIONS MONITOR REQUIREMENTS (OPTIONAL)

In programs requiring multiple communications output, i.e. transmission of particular reply messages to more than one terminal, the user's Communications Monitor routine will require a table of device numbers in order to initiate output to each terminal in turn. Communications requests specifying multiple output, if issued in the recommended format (see Appendix 1), assume that such a table exists.

The user can set up this table either within the Master routine or within Communications Monitor itself. Further details are given in the description of Communications Monitor in Chapter page 32.

ACTIVATING THE DRIVER

The following are three possible methods of making the initial entry to Driver from the Master routine.

1 If an Initialization bead is in use, preset Words 0, 10 and 12 of the TAB to request entry to this bead, as follows

Word 0 = 0 (request code 0000)

Word 10 = 0

Word 12 = Bead number of Initialization bead

Control is passed to Driver in the same manner as a PLAN bead, that is, on the instruction

BRN HDRRACUE

2 If no Initialization bead is in use, but a start-of-day output message is to be broadcast then:

(a) Set Words 0 to 7 of the TAB to request communications output and deallocation of the TAB

(b) Set up the start-of-day message when defining the TAB's Message area.

(c) Pass control to Driver as above.

The program will output the specified message and suspend awaiting input of the first user message.

3 Using a suitable entry point, branch directly to the suspension procedure, or any prior initialization procedures, within Communications Monitor.

PROGRAMMING PROCEDURE - MULTITHREADING PROGRAMS

System considerations

The standards given for single threading programs on pages 2 to 4 apply equally to multithreading programs, with the exception of the Driver routines to be listed under the #PERMANENT directive. The routines selected for inclusion

in the program should be listed using the names indicated below.

Routine	*Name*
Request Analyser 2	HDRRA2
Request Analyser 3	HDRRA3
System Control	HDRSC2
Bead Scheduler 2	HDRBS2
Bead Scheduler 3	HDRBS3
Bead Scheduler 4	HDRBS4
Bead Scheduler 5	HDRBS5
Store Administrator	HDRSA2
Peripheral Monitor:	
P.M. Entry	HDRPMENTRY
P.M. Queue-select	HDRPMQSELEC
P.M. Cycle	HDRPMCYCLE
P.M. Continuation	HDRPMCONT
P.M. Exit	HDRPMEXIT
Communications Monitor:	
C.M. Entry/Exit	HDRCM1
C.M. 7900 Interface	HDRCMA
C.M. Multiplexor interface	HDRCMB
C.M. 7900 queue selector	HDRCMQS1
C.M. Multiplexor queue-selector	HDRCHQS2
Subroutines:	
Overlay control	HDROC2
Deallocate TAB	HDRDTAB
Enter Exception Mode	HDREEM
Return to Normal Mode	HDRNORM

Routine	*Name*
Queueing subroutine 1	HDRATQ
Queueing subroutine 2	HDRRFQ
Queueing subroutine 3	HDRAHQ
General suspension 1	HDRGS1
General suspension 2	HDRGS2

Main store organisation

TAB AREA DEFINITIONS

The number of TABs defined in the Master routine must be equal to the maximum number of messages to be handled by the program. All but one of these TABs must be organised to form the initial free TAB queue: the remaining TAB must be set to request the first entry to Driver.

All TABs should be defined under a #UPPER directive. The following standards must be observed in order to set up the free TAB queue.

1 Word 21 of each TAB in the queue must be set to contain the start address of the TAB following it. Word 21 of the last TAB in the queue must be set zero, indicating end- of-queue.

2 A 2-word queue management area must be defined for the queue under a #UPPER directive. Word 0 of the area must be set to contain the start address of the first TAB in the queue, and word 1 set to contain the start address of the last TAB in the queue.

3 The start address of the queue management area must be specified in word 0 of the 2-word common block HDRAFTQ under a #LOWER directive. Word 1 of HDRAFTQ must be

set negative.

The remaining TAB should be set for initial entry to Driver using either method 1 or (fixed store systems only) method 2 as described in the section *Activating the Driver*, pages 16 and 17. Word 21 of this TAB should be set zero. Note that where output of a start-of-day message is required under a dynamic store system, the functions of setting up the message and issuing the appropriate output macro should be performed within the initalization bead.

A number of additional standards apply to the individual TABs, as follows.

1 Under a fixed store system, words 22 to 24 of each TAB should be set as for a single threading program. That is, they should respectively contain the start addresses of the Message, Input/Output and Additional Core areas associated with the TAB. Under a dynamic store system, word 22 must similarly contain the start address of the Message area, but words 23 and 24 should be set zero.

2 Words 25 and 26 (permanent and temporary processing priorities) must be set zero, irrespective of the type of queueing system in use.

3 Word 27 of all TABs must be set to contain the start addresses of the free TAB queue.

STORE CELL COMMON POOL (DYNAMIC STORE SYSTEMS ONLY)

Each block in the common pool must be specified as a suitable number of equal sized cells chained together to form a queue. For the purpose of issuing store requests, the blocks are taken to be numbered in the order in which they are set up: the first block specified will be referenced in store

requests as block 0, the second as block 1, and so on. Store cell blocks may be declared under a #LOWER or #UPPER directive.

The following standards must be observed:

1 Within each block, words 0 to 3 of each cell must be set as follows:

Word 0 Total length of the cell in words

Word 1 Start address of the next cell in the block (zero in the case of the last cell in the block)

Word 2 Zero

Word 3 Start address of the queue management area of the block (see 3 below). In practise it will probably be necessary to set word 3 of each cell zero when specifying the blocks and include coding to load the appropriate addresses after the #ENTRY directive.

2 Three 2-word queue management areas are required by Store Administrator for each block in the pool. These are specified as a series of 6-word entries in a table with the common blockname HDRSAQ, starting with the entry for block 0 and continuing in ascending sequence of block number:

The entry for each block is as follows:

Word 0 Start address of first cell in block

Word 1 Start address of last cell in block

Words 2 to 5 Zero

Where words 0 and 1 are the queue management area of the block and words 2 to 5 are the queue management areas of the first and second TAB queues for that block (see Chapter 6, page).

HDRSAQ must be terminated by a word set negative.

Example

The following coding will set up a common pool consisting of two 500 word cells and four 64-word cells.

```
#UPPER
STOR0     500,0/STOR1+500,0,0
          DATA0(496)
STOR1     500,0,0,0
          DATA1(496)
STOR2     64,0/STOR2+64,0,0
          DATA2(60)
STOR3     64,0/STOR3+64,0,0
          DATA3(60)
STOR4     64,0/STOR4+64,0,0
          DATA4(60)
STOR5     64,0,0,0
          DATA5(60)
#LOWER         COMMON/HDRSAQ/
SAQ            0/STOR0,0/STOR1,0,0,0,0
               0/STOR2,0/STOR5,0,0,0,0
               -1
               .
               .
               .
               .
               .
               .
```

Block 0 (two 500 - word cells)

Block 1 (Four 64- word cells)

```
                .
                .
                .
#ENTRY          .
                .
                .
                .
                .
                .
                LDN   3   SAQ          [Get address of Block 0 QMA
NXTBLK          LDX   2   0(3)         [Get address of first cell
                STO   3   3(2)         [Store QMA address in word 2 of cell
                LDX   2   1(2)         [Get address of next cell in block
                BNZ   2   *-2
                ADN   3   6            [Get address of QMA for next block
                BPZ   3   NXTBLK
```

OTHER USER DEFINED LOCATIONS AND AREAS

The following are all declared as common blocks, under a #LOWER directive unless otherwise stated

Bead number of initial bead (HDRBDA)

Whenever Communications Monitor allocates a TAB to an input message, it will set word 12 of the TAB to contain the bead number held in HDRBDA. The bead whose number is held in HDRBDA will hence always be the first bead entered in the course of processing any message.

Length

1 word

Setting

Bead number of Initial bead (conventionally 1)

Highest bead number constant (HDRBDCONST)
Set as for the single threading driver. See page 10.

Queue management area for Bead Scheduler internal queues (HDRBDQ)

For programs employing a bead priority queueing system (Bead Scheduler 3 or 5) a queue management area is required for each internal queue (one queue of TABs per bead). The areas are declared contiguously under the common blockname HDRBDQ.

Length

(2 x number of beads) + 1 words

Setting

The queue management areas should all be set to zero and the block terminated with a word set negative.

Queue management area for Bead Scheduler External queue (HDRBSEXTQ)

Under all queueing systems, a queue management area is required for the Bead Scheduler external queue.

Length

2 words

Setting

Zero

Queue management area for Bead Scheduler internal queue (HDRBSINTQ)

In programs employing the standard first-in-first-out queueing system (Bead Scheduler 2 or 4) a queue management area is required for the single TAB internal TAB queue maintained by Bead Scheduler under this system.

Length

2 words

Setting

Zero

Bead Scheduler user entry point (HDRBSUE)

This enables a user-written subroutine to be called in order to implement message priority queueing within Bead Scheduler. Message priority queueing cannot be combined with bead priority queueing, and the entry point may therefore only be used in programs incorporating Bead Scheduler 2 or 4. Depending in which of these versions is in use the instruction in HDRBSSUE will be OBEYed either after the routine adds a TAB to its external queue (Bead Scheduler 2) or is about to add a TAB to its internal queue (Bead Scheduler 4). Full details are given in the section *Message Priority queueing* in Chapter 6.

Length

1 word

Setting

Where first-in first-out queueing is required:

```
NULL                (Bead Scheduler 2)

CALL  1  HDRATQ     (Bead Scheduler 4)
```

Otherwise:

```
CALL  1  username
```

where *username* is the cue name/entry point of the user's subroutine.

Bead Branch table (HDRBT1)

Set as for the single threading Driver. See page 10.

Bead Storage table (HDRBT2)

Set as for the single threading Driver. See page 14.

Bead free/busy indicator table (HDRBT3)

This table contains a 1-word entry for each bead referenced in HDRBT1 and HDRBT2. Each time control is passed to a bead, Bead Scheduler will set an indicator in the appropriate table entry. This indicator will remain set until final exit from the bead, and will prevent servicing of any further requests for transfer of control to that bead until it has completed its execution in respect of the current TAB.

Length

(number of beads)+1 words

Setting

The entries for the beads must be set to zero. The table must be terminated with a word entry set negative.

Communications Monitor closedown indicator (HDRCMAFIN)

This indicator will be set by whichever C.M. interface routine is in use when it becomes necessary to inform the CM Master routine that closedown of the Communications network has occurred.

Length

1 word

Setting

Zero

Communications Monitor GET constants (HDRCMAGC)

HDRMAGC contains the constants to be held in words 1 and 2 of the users control area HMPUC whenever the Communications Monitor GET routine obtains input (MPGET macro:- See the appropriate chapter of *Data communications and interrogation*, edition 2 TP 4201).

Length

2 words

Setting

Word 0 Maximum message length in characters

Word 1 Message terminator character

Communications Monitor input/output subroutine calls (HDRCMAGET, HDRCMAPUT)

These are only required by program using the multiplexor housekeeping package. They allow the user to choose between the various input and output subroutines available in the multiplexor package.

Length

1 word each

Setting

For input, HDRCMAGET may be set to contain any one of the following instructions:

```
CALL  1  HMPGET

CALL  1  HMPGETTRANS

CALL  1  HMPSEGGET
```

For output, HDRCMAPUT may be set to either

```
CALL  1  HMPPUT
```

or

```
CALL  1  HMPPUTTRANS
```

A full description of these subroutines is given in the manual *Data communications and interrogation*, (edition 2 TP 4201) Chapter 8, pages 151 and 154.

Queue management areas for communications Monitor output queues (HDRCMAOQ)

A queue management area is required for each output queue (one queue per teleprocessor). The areas are declared contiguously under the common blockname HDRMAOQ.

Length

(2 x number of teleprocessors) + 1 words

Setting

The queue management areas should all be set to zero. The block must be terminated with a word set negative.

Communications Monitor user entry point 1(HDRCMAQC1)

The instruction held in this location is OBEYed by Communications Monitor whenever a TAB is to be placed on the Manipulate queue (see Chapter 6 page 25)

Length

1 word

Setting

Where TABs are to be queued by the standard first-in first-out method, HDRCMAQC1 should be set to contain a call to standard queueing subroutine 1, that is, the instruction

```
CALL  0  HDRATQ
```

If, instead, a user-written subroutine is to be called at this point, HDRCMAQC1 should be set to contain the instruction

```
CALL 0  username
```

where username is the cue name/entry point of the subroutine.

Communications monitor user entry point 2 (HDRCMAQC2)

The instruction held in this location is OBEYed whenever a TAB is to be placed on one of the output queues.

Length

1 word

Setting

Where TABs are to be queued by the standard first-in first-out method, HDRMAQC2 should be set to contain a call to standard queueing subroutine 1, that is, the instruction

```
CALL  0  HDRATQ
```

If, instead, a user written subroutine is to be called at this point, HDRMAQC2 should be set to contain the instruction

```
CALL  0  username
```

where username is the cue name/entry point of the subroutine.

Communications monitor TEST constant (HDRCMATES)

HDRCMATES contains the user parameter required by the MPTES macro within the Test for Exceptions routine. It determines which console messages will be displayed by the housekeeping.

Length

1 word

Setting

0 or 1 (See the section *The macro MPTES* in the appropriate Chapter of *Data Communications and Interrogation* (Edition 2 TP 4201).

Additional user entry points - communications monitor PUT and GET routines (HDRCMAUC1, HDRCMAUC2, HDRCMAUC3)

These entry points enable user subroutines to be called in order to analyse reply information after message input and output (HDRCMAUC1 and HDRCMAUC3 respectively) and manipulate control information before output (HDRCMAUC2).

HDRCMAUC1 enables a call to be made to a user-written subroutine immediately after input by the GET routine (MPGET macro). It is thus possible to analyse the contents of the reply area HMPHR at this point and initiate recovery from transient errors without sending the program into exception mode.

HDRCMAVC2 similarly enables the contents of the control area HMPUC to be manipulated by a user subroutine immediately before output of a message by the PUT routine (MPPUT macro).

HDRCMAUC3 provides the same facility as HDRCMAUC1, in this case after output of a message by the PUT routine.

For details of the information held in HMPHR and HMPUC at these points in the execution of Communications Monitor, see the description of the MPGET and MPPUT macros in the appropriate chapter of *Data communications and interrogation*,

edition 2 ,TP4201.

Length

1 word each

Setting

Where a user subroutine is to be called:

CALL 0 *username*

where *username* is the cue name/entry point of the subroutine. Otherwise NULL

Communications Monitor input displacement constant (HDRCMAUID)

This location will be accessed by Communications Monitor before storing an input message in the Message area of the TAB currently heading the free TAB queue. The value held in HDRCMAUID specifies the address (relative to word 0 of each message area) at which storage of input messages is to begin.

Length

1 word

Setting

≥5. (Words 0 to 4 of all Message areas are reserved for Driver parameters).

Queue management areas - Communications Monitor Wait queues (HDRCMAWQ)

Where communications input and output is to be performed using a message buffering system, a 2-word queue management area is required for each Wait queue (one queue of TABs for teleprocessor). The areas are specified contiguously to form the common block HDRCMAWQ.

Length

(2 x number of teleprocessors) + 1 words

Setting

The queue management areas should all be set to zero. The block should be terminated with a word set negative.

Operator intervention constant (HDRCMO1)

Each time it is entered, Communications Monitor will examine the contents of word 30 to determine whether operator intervention has occurred. This is done by performing a logical AND operation on the contents of word 30 and the contents of HDRCMO1.

Length

1 word

Setting

The bits within word 30 that are to be tested must be set in HDRCMO1. For example, if HDRCMO1 is set equal to #5, Communications Monitor will test bits 21 and 23 of word 30, If this facility is not required, HDRCMO1 shoudl be set to zero.

Communications Monitor table 1 (HDRCMTAB1)

A CALL instruction, held as a one-word entry in this table, is OBEYed by communications Monitor in order to call the interface routine appropriate to the type of housekeeping in use.

Length

7900 housekeeping: 2 words
Multiplexor housekeeping: 3 words

Setting

For 7900 housekeeping:

Word 0 CALL 1 HDRCMA

Word 1 Set negative

For multiplexor housekeeping:

Word 0 Zero

Word 1 CALL 1 HDRSCMB

Word 2 Set negative.

Communications Monitor table 2 (HDRCMTAB2)

A CALL instruction, held as a one-word entry in this table, is OBEYed by Communications Monitor in order to call the queue - selector routine appropriate to the type of housekeeping in use.

Length

7900 housekeeping: 2 words
Multiplexor housekeeping: 3 words

Setting

For 7900 housekeeping:

Word 0 CALL 1 HDRCMQSI

Word 1 Set negative

For Multiplexor housekeeping:

Word 0 Zero

Word 1 CALL 1 HDRCMQS2

Word 2 Set negative

Driver count (HDRDCT)

The setting of HDRDCT determines the frequency with which full scans occur during the operation of the program.

Length

2 words

Setting

Both words should be set to the number of entries to Driver per full scan. For example, the value 4 in each word will cause a full scan to occur on every fourth entry to Driver.

Dynamic store indicator (HDRDCIND)

This indicator informs Driver whether a fixed or dynamic store system is in use.

Length

1 word

Setting

0 for a fixed store system
1 for a dynamic store system.

FHR branch table (HDRFHRCALL)

This table contains a branch to each file handling routine. On receipt of a request for peripheral input or output, Peripheral Monitor will branch to the required routine by OBEYing the appropriate instruction.

Length

(number of file handling routines) + 1 words

Setting

As recommended for the single threading Driver. See Chapter 5 page 27.

FHR exception mode indicator (HDRFHREMI)

This location will be set by Peripheral Monitor when it becomes necessary to inform the file handling routines that the program has entered exception mode.

Length

1 word

Setting

Zero.

FHR error reply indicator (HDRFHRERI)

Whenever a file handling routine detects an error in an input or output transfer, it will set this indicator with the appropriate error code before re-entry to Peripheral Monitor. Peripheral Monitor will set word 0 of the service TAB and word 8 of the TAB accordingly. The procedure involved is similar to that described for the single threading version of this routine in Chapter 5, page 27.

Length

1 word

Setting

Zero.

Device reply word addresses (HDRFHRREP)

This table contains a 1-word entry for each file handling routine.

Length

(number of file handling routines) + 1 words

Setting

The entries for the file handling routines must be declared in ascending order of file reference number.

Where file handling is to be performed at PERI level, the entry for each routine should be set to contain the address of word 1 of the control area. If a routine uses more than one control area, its entry should be set to zero.

Where peripheral input and output is to be performed by housekeeping, the entry for each routine must be set to contain the address of the user's reply indicator for that routine (see file handling routines, Chapter 6, page 36).

In either case, the table must be terminated by a word set negative.

Queue management areas for Peripheral Monitor device queues (HDRFHRQ)

Under all queueing systems, a two-word queue management area is required for each device queue. The areas are set up contiguously to form the common block HDRFHRQ.

Length

(2 x number of device queues) + 1 words

Setting

The queue management areas should all be set to zero. The block must be terminated by a word set negative.

7900 Housekeeping indicators (HDRHMPINDS)

In programs using 7900 housekeeping, the macro MPLA must be issued under the common blockname HDRHMPINDS in order to establish the housekeeping lower data areas.

Length

(3 x number of teleprocessors) + 24 words

Setting

HDRHMPINDS is set by issuing the macro MPLA.

Lower user common area (HDRLUCDAT)

Set up as for the single threading Driver. See Chapter 4, page 9.

Overlay indexes (HDROLIND1, HDROLIND2)

For programs using the standard Driver overlay system (that is, incorporating Request Analyser 3 and Bead Scheduler 3 or 5) two tables are required, each containing a 1-word entry for each area of overlay up to the highest numbered area in use.

Length

(Highest area number +2) words per table

Setting

For each overlay area in use, the entry in HDROLIND1 must be set to contain the start address of the corresponding group of 5-word entries in the overlay area table HDROLTAB (see below). Within HDROLIND2, the entry for each area must specify the number of units in that area. Since area numbers begin at 1, each table must commence with a vacant word so that the area numbers can be used as modifiers. Both tables must be terminated with a 1-word entry see negative.

For example, a program using 5 overlay areas would require the following entries in HDROLIND1 and HDROLIND2.

	HDROLIND1	*HDROLIND2*
Word 0	0	0
Word 1	0/HDROLTAB	*a*
Word 2	0/HDROLTAB+5*a*	*b*
Word 3	0/HDROLTAB+5*(a+b)*	*c*
Word 4	0/HDROLTAB+5*(a+b+c)*	*d*
Word 5	0/HDROLTAB+5*(a+b+c+d)*	*e*
Word 6	-1	-1

where *a*,*b*,*c*,*d* and *e* are the number of units in areas 1,2,3, 4 and 5 respectively.

Overlay area table (HDROLTAB)

This table contains a 5-word entry for each unit of overlay. Request Analyser and Bead Scheduler use the table to schedule the overlaying of beads into main store, the entries for eac

overlay area being accessed by means of the appropriate link address in HDROLIND1 and counter/modifier in HDROLIND2 (See above).

Length

(5 x number of overlay beads) +1 words

Setting

The table must appear under a #UPPER directive. The entries for the overlay units should all be set to zero and terminated by a 1-word entry set negative.

Overlay count and exception indicators (HDROLCT, HDROLX)

These are Driver working locations, used only in overlay programs.

Length

1 word each

Setting

Zero

Highest file reference number constant (HDRPMCONST)

This location is used by Peripheral Monitor to check that a valid file reference number has been specified in each request for peripheral input or output.

Length

1 word

Setting

Highest file reference number +1

Peripheral Monitor user entry point 1 (HDRPMENT1)

The instruction held in this location is OBEYed by Peripheral Monitor immediately before the point at which control is to be passed to the standard queue-selector routine.

Length

1 word

Setting

Where a user written queue selector routine is to be substituted for a standard version, HDRPMENT1 must be set to contain the instruction

CALL 0 *username*

where *username* is the cue name/entry point of the user's queue selector routine.

Where this facility is not required, HDRPMENT1 should be set to contain the instruction NULL.

Peripheral Monitor user entry point 2 (HDRPMENT2)

The instruction held in this location is OBEYed by the standard Peripheral Monitor queue-selector whenever a TAB is to be placed in one of the device queues.

Length

1 word

Setting

Where TABs are to be queued by the standard first-in first-out method, HDRPMENT2 should be set to contain a call to standard queueing subroutine 1, that is, the instruction

```
CALL  0  HDRATQ
```

If, instead, a user-written subroutine is to be called at this point, HDRPMENT2 should be set to contain the instruction

```
CALL  0  username
```

where *username* is the cue name/entry point of the user's subroutine.

System control first branch table (HDRSCBR1)

This table consists of CALLs to Peripheral Monitor, Communications Monitor, Bead Scheduler and (if used) Store Administrator. The instructions are stored in the order in which the routines are to be entered during a full scan.

Length

Either 3 or 4 words, depending on whether store Administrator

is present in the program.

Setting

The optimum order of entry will depend on the characteristics of the individual system. Typical sequences are as follows:

For a fixed store system:

Word 0 CALL 1 HDRCMCUE

Word 1 CALL 1 HDRPMCUE

Word 2 CALL 1 HDRBSCUE

For a dynamic store system:

Word 0 CALL 1 HDRCMCUE

Word 1 CALL 1 HDRPMCUE

Word 2 CALL 1 HDRSACUE

Word 3 CALL 1 HDRBSCUE

The table must always terminate with the call to Bead Scheduler, as shown above.

System control second branch table (HDRSCBR2)

System control uses this table to branch to the required Driver routine in response to a specific request (as appeared to initiating a full scan).

Length

3 or 4 words, depending on whether Store Administrator is present in the program

Setting

Word 0 CALL 1 HDRBSCUE

Word 1 CALL 1 HDRPMCUE

Word 2 CALL 1 HDRCMCUE

Word 3 CALL 1 HDRSACUE (dynamic store systems only).

The service TAB (HDRSCST)

Use of HDRSCST by Driver is described in Chapter 2, page 17.

Length

3 words

Setting

Zero

TAB count (HDRTABNO)

If the General Suspension routine detects that the communication network has been closed down, the number of TABS on the free tab queue will be compared with the value in HDRTABNO. If the two values are found to be equal, indicating that there is no further processing to be done by the program, the General Suspension routine will

branch to the users end point (HDREND) is the Master Routine.

Current TAB's address (HDRTABSTORE)

Use of HDRTABSTORE by Driver and the beads is described in Chapter 2, page 17.

Length

2 words

Setting

Word 0 start address of the TAB to be used for initial entry to Driver.

Word 1 Zero.

Enter Exception Mode subroutine user entry point (HDRUENT)

If Driver detects an error whilst the program is already in exception mode, the Enter exception mode subroutine will OBEY the instruction held in HDRUENT and then branch to the multiple error halt in HDRMEHALT (below).

Length

1 word

Setting

A CALL or BRN to a user routine, or NULL if this facility is not required.

Upper user common area (HDRUUCDAT)

Set as for the single threading Driver. See page 9.

USER END POINTS

Two user end points, HDRMEHALT and HDREND, must be included under a #CUE directive.

Driver will branch to HDRMEHALT following detection of a multiple error. It will branch to HDREND when the general suspension subroutine detects that no further processing is possible. The user coding associated with each end point may be limited to a single SUSWT instruction, or alternatively consist of restart or postmortem routines.

ENTRY TO DRIVER

The multithreading Driver is entered in exactly the same manner as the singlethreading version, that is on the instruction

```
BRN HDRRACUE
```

However, if the Overlay control routine is present in member 2, it must have previously been activiated by the instruction

```
AUTO 2 HDRRCUE
```

Similarly, if a user-written file handling routine incorporating housekeeping is present in member 3, this member must also have been previously activated on the instruction

Auto 3 *cuename*

where *cuename* is the cuename/entry point of the user's file handling routine.

Chapter 5

Single-threading Driver routines

INTRODUCTION

The control element of a single threading program, otherwise known as the single threading Driver, comprises 1900 Driver routines supplied by ICL as standard software, together with further control routines written by the user.

The 1900 Driver Control routines consist of:

Request Analyser

System Control

Bead Scheduler

User written control routines consist of:

Peripheral Monitor (and, optionally, associated file handling routines). In the unlikely event of no peripheral input/output being required by the program, this routine can, of course, be omitted.

Communications monitor

Store Administrator (optional dummy routine)

Any additional control routines written by the user to meet special requirements.

When a PLAN bead issues a request these routines will normally be entered in the order indicated below:

Following a bead request:

1 Request Analyser

2 System Control

3 Bead Scheduler

4 Next Bead

Following any other type of request:

1 Request Analyser

2 System Control

3 Either:

(a) Peripheral Monitor (following a peripheral request)

or (b) Communications Monitor (following a communications request)

or (c) Store Administrator (following a store request)

or (d) Any one of any additional control routines supplied by the user, following the appropriate request

4 System Control

5 Bead Scheduler

6 Next bead

A COBOL bead issuing a request exits to Driver using an entry point provided within Bead Scheduler. The control

routines are then entered in the order given above.

The service TAB

The Service TAB is a 3 word parameter area which is held starting at the location HDRSCST. In the single threading Driver it is manipulated by all those routines which exit to System Control, namely Request Analyser and all user-written control routines.

The service TAB is used by these routines in the following manner:

1 Word 0 is set by Request Analyser and all user-written control routines before entry to System Control. This word is subsequently examined by Bead Scheduler to determine whether servicing of the current request has been successful. If an error has occurred, Bead Scheduler will pass control to the Error Recovery bead instead of the bead specified in the request. Bead Scheduler also sets this word if it fails to service a bead request, and hence indicates to itself that control is to be passed to the Error Recovery bead.

2 Word 1 is set by Request Analyser only, and is used by System Control to determine which control routine is to be entered to service the current request.

Word 2 of the service TAB is of no significance in a single threading Driver.

1900 DRIVER CONTROL ROUTINES

Specifications of the three control routines included in the single threading 1900 Driver package are given in this section. It is possible that the user may wish to modify or replace some of these routines to meet special requirements, and their operation is hence described in sufficient detail to enable him to do so.

HDRRA1

NAME

HDRRA1

TITLE

Request Analyser

ENTRY POINT/CUE NAME

HDRRACUE

DESCRIPTION

Request Analyser initiates servicing of all requests issued by the beads in the program. PLAN beads branch directly to the routine: entry from COBOL beads is via Bead Scheduler.

The actions of the Request Analyser are described below in order of execution.

1 STORE ACCUMULATORS

The routine determines the location of the TAB in main store using the address held in HDRTABSTORE. The contents of accumulators 4 to 7 and 0 are stored in words 16 to 20 of the TAB respectively.

If this facility is not required, the user must code his own Request Analyser in accordance with the standards given

in the section *Control* below.

2 ANALYSE THE CURRENT REQUEST

The facility code (i.e. the first character of the request code in word 0 of the TAB) is checked to ensure that it is less than the facility code constant held in the location HDRRACONST.

If the code is invalid, the Service TAB HDRSCST is set thus:

Word 0 3

Word 1 0

Word 2 0

Word 8 of the TAB (error reply parameter) is set to zero.

If the code is valid, the required settings of HDRSCST are

Word 0 1

Word 1 *f*

Word 2 0

where *f* is the facility code within the request code just validated, held in decimal form.

3 BRANCH TO SYSTEM CONTROL

Request Analyser branches to System Control on the instruction

```
BRN HDRRSCUE
```

CONTROL

Entry

Request Analyser is entered from:

1 The Master routine (on initial entry to Driver only)

2 All beads written in PLAN

3 Bead Scheduler (following entry to Bead Scheduler from a COBOL bead)

Each of the above routines branch to Request Analyser on the instruction

```
BRN HDRRACUE
```

Exit

The routine branches to System Control on the instruction

```
BRN HDRSCCUE
```

Locations used	*Notes*
HDRTABSTORE HDRRACONST	Preset in Master routine. See *Request Analyser requirements*, Chapter 4, page 9.
The TAB (words 0 and 17 to 20 only)	Preset for entry from Master routine (see *TAB presets*, Chapter 4, page 9). Subsequently set by (a) PLAN beads before branching to Request Analyser (b) COBOL beads before EXIT to Bead Scheduler See *Issuing requests*, Chapter 3, pages 8 to 16.
HDRSCST (Service TAB)	Settings on entry to Request Analyser are irrelevant. For settings on exit to System Control see *Description*, above.

HDRSC1

NAME

HDRSC1

TITLE

System Control

ENTRY POINT/CUE NAME

HDRSCCUE

DESCRIPTION

System Control is entered from Request Analyser when the latter routine has completed its contribution to the servicing of a request. The routine interprets the parameters set up in the Service TAB by Request Analyser and accordingly branches to the next control routine required to service the request.

If Request Analyser has detected an error, or the current (valid) request is for transfer of control to another bead, System Control branches to Bead Scheduler and makes no further contribution to the servicing of the request. Otherwise, System Control branches to any one of the following control routines, as required

Peripheral Monitor

Communications Monitor

Store Administrator

On completion of execution, the selected routine sets the Service TAB to indicate whether successful or unsuccessful processing has occurred, and branches back to System Control. System Control then branches to Bead Scheduler, thus enabling Bead Scheduler to analyse the parameters set up in the Service TAB and pass Control to the next bead.

The actions of the routine are described below in order of execution.

1 LOOK UP MAIN BRANCH TABLE

The routine OBEYS one of 3 instructions held in a 4-word Main Branch Table (HDRSCMB). The appropriate instruction in HDRSCMB is located and executed using an OBEY instruction modified by the contents of word 0 of the Service TAB as set by the entering routine. The possible settings of this word on entry to System Control are as follows.

1 On entry from Request Analyser

	Service TAB setting
Valid request:	Word 0 = 1
Invalid request:	Word 0 = 3

2 On entry from any other routine

Following successful processing:	Word 0 = 2
Following unsuccessful processing:	Word 0 = 3

The instructions in the Main Branch Table are as follows:

Word 0 Not applicable to single threading Driver

Word 1 Modified OBEY on the Second Branch Table HDRSCBR2 (see below). It follows from the Service TAB settings given above that this instruction can only be executed following entry from Request Analyser.

Word 2 BRN HDRBSCUE (branch to Bead Scheduler following successful processing)

Word 3 BRN HDRBSCUE (branch to Bead Scheduler following detection of an invalid request or unsuccessful processing by the entering routine).

2 (a) BRANCH TO REQUIRED CONTROL ROUTINE

If System Control has been entered from Request Analyser and the current request is valid, the routine accesses the second Branch Table HDRSCBR2 in order to locate and execute the branch instruction to the required control routine. This is done by means of the OBEY instruction in word 1 of the Main Branch Table, which is modified by the value of the facility code held in word 1 of the Service TAB.

The branch instructions appear in 4 consecutive words of the Second Branch Table in the following order

Word 0	BRN HDRBSCUE	(Branch to Bead Scheduler)
Word 1	BRN HDRPMCUE	(Branch to Peripheral Monitor)
Word 2	BRN HDRCMCUE	(Branch to Communications Monitor)
Word 3	BRN HDRSACUE	(Branch to Store Administrator

System Control will thus OBEY the branch instruction appropriate to the current facility code. For example, if the current code specifies communications input or output (facility code 2) the branch instruction in word 2 of the table (branch to Communications Monitor) will be OBEYed.

(b) BRANCH TO BEAD SCHEDULER

Words 2 and 3 of the Main Branch Table both contain the instruction for a branch to Bead Scheduler, i.e.

```
BRN HDRBSCUE
```

Peripheral Monitor, Communications Monitor and Store Administrator all report successful servicing of a request by setting word 0 of the service TAB to 2 before branching to System Control. The branch instruction in word 2 of the Main Branch Table is therefore OBEYed and Bead Scheduler entered to initiate processing by the next bead.

Request Analyser and all the above control routines report errors by setting word 0 of the service TAB to 3 before branching to System Control. The identical branch instruction in word 3 of the Main Branch Table is thus OBEYed and Bead Scheduler entered to initiate error recovery by the appropriate bead.

Note that in both of the above cases, System Control takes no action other than branching to Bead Scheduler. Bead Scheduler will allow further processing or initiate error recovery according to the setting of word 0 of the Service TAB, which is not manipulated by System Control.

CONTROL

Entry

System Control is entered from

1 Request Analyser

2 Peripheral Monitor

3 Communications Monitor

4 Store Administrator

Each of the above routines completes its execution by branching to System Control on the instruction

BRN HDRSCCUE

Exit

The routine branches to

1	Peripheral Monitor	(BRN HDRPMCUE)
2	Communications Monitro	(BRN HDRCMCUE)
3	Store Administrator	(BRN HDRSACUE)
4	Bead Scheduler	(BRN HDRBSCUE)

Note: System Control will also exit to, and be re-entered from, non-standard user-written control routines if suitable modifications are made to the Second Branch Table. See page 21 and also *System Control Requirements*, Chapter 4.

Locations used

	Notes
HDRSCMB HDRSCBR2	Preset within System Control
HDRSCST (Service TAB)	

NAME

HDRBS1

TITLE

Bead Scheduler

ENTRY POINT/CUE NAME

HDRBSCUE

DESCRIPTION

Bead Scheduler is the last routine entered in the course of servicing a request and causes control to be passed to the next bead required to continue processing of the current message. The routine works on the following assumptions, which will always be true in a single threading program:

1 All beads in the program are free to carry out processing.

2 Only one TAB is present in the program, and the start address of the TAB is held in the location HDRTABSTORE.

The routine uses the Bead Storage Table HDRBT2 and Bead Branch Table HDRBT1 set up by the user in his Master routine. Depending on the type of entry Bead Scheduler uses the contents of word 11 or 12 of the TAB as a modifier in order to access the appropriate item in each of these tables.

Bead Scheduler determines the type of entry according to the setting of word 0 of the Service TAB and acts accordingly as follows:

	Service TAB setting	
1	Word 0 = 3	An invalid request code has been detected; or an error has occurred during servicing of the current request by one of the control routines. Bead Scheduler sets accumulator 1 to zero and then carries out an OBEY instruction on the Bead Branch Table using this value as a modifier. The first CALL instruction in the table (always to the Error Recovery Bead) is hence OBEYed.
2	Word 0 = 1	The current request is for transfer of control to another bead only (i.e. Peripheral Monitor, Communications Monitor or Store Administrator have not been executed). The only valid request code in this case is 0000. The request code is validated. If it is invalid, word 8 of the TAB (error reply word) is set to 0001 and action is taken as in 1 above. If it is valid, action is taken as in 3 below.

Service TAB setting

3	Word 0 = 2	The request issued to Driver has been serviced and a bead is now to be entered to continue processing. This involves either entry to another bead, or re-entry to the bead which issued the request.

The subsequent actions taken by Bead Scheduler are described below in order of execution.

(a) Identify required bead

Word 10 of the TAB is checked to see if it is zero or non-zero.

If word 10 is zero, this indicates that control is to be passed to another bead. The bead number of the bead to be entered is found in word 12 of the TAB. This number is subsequently placed in word 11 of the TAB following validation (see (b) below).

If word 10 is not zero (i.e. contains a bead-generated re-entry point parameter), the bead which issued the request is to be re-entered and its bead number is found in word 11 of the TAB.

(b) Check for valid bead number

The only vet on the identity of the bead to be entered is that the specified bead number must be less than the value specified by the user (in the location HDRBDCONST) in the Master routine.

If the bead number is found to be invalid, word 8 of

the TAB is set to 0002 and word 0 of the service TAB is set to 3. Bead Scheduler will then initiate error recovery as described in 1 above. If the bead number is valid, processing continues as described below.

(c) Pass control to required bead

The bead number of the bead to be entered, which by this stage is always held in word 11 of the TAB, is used as a modifier in order to access the appropriate entries in the bead storage and bead branch tables.

The contents of the appropriate entry in the bead storage table are first examined. If this is zero, it indicates that the required bead is resident in main store and that creating is therefore not required. If the entry contains an area/unit number, this is used as a parameter in a call to the overlay routine %EROL and the specified overlay bead is called in.

The contents of accumulators 4 to 7 and 0 are then restored from words 16 to 20 of the TAB and the appropriate CALL instruction in the Bead Branch Table is OBEYed, thus causing the required bead to be entered.

The OBEY instruction is followed by 5 further instructions to meet the requirements of COBOL beads. The first 6 instructions load the following addresses into accumulator 3

1 Start address of TAB

2 Start address of Message area

3 Start address of Input/Output area (if applicabl

4 Start address of additional core area (if applicable)

5 Start address of Lower User Common Area

6 Start address of Upper User Common Area

These are used as parameters by the bead being entered if it is written in COBOL.

The seventh instruction is the point of return from a COBOL bead and is thus a branch to Request Analyser, that is,

BRN HDRRACUE

CONTROL

Entry

System Control branches to Bead Scheduler on the instruction

BRN HDRBSCUE

Exit

Bead Scheduler branches to the required bead by means of the appropriate CALL instruction in the user's Bead Branch Table HDRBT1 (see Chapter 4, page 10).

Locations used

	Notes
HDRTABSTORE	Set up in the Master routine -
HDRBDCONST	see *Bead Scheduler requirements*,
HDRBT1	Chapter 4, page 10).
HDRBT2	

Notes

HDRSCST (Service TAB)

The TAB

USER-WRITTEN CONTROL ROUTINES - INTRODUCTION

User written control routines must conform to the following standards

1 All routines, with the exception of any file handling routines, are entered from, and exit to, System Control. File handling routines are entered from, and exit to, Peripheral Monitor.

It follows that Peripheral Monitor, Communications Monitor and Store Administrator must be assigned standard cue names in order to be entered from System Control. These are:

HDRPMCUE	(Peripheral Monitor)
HDRMMCUE	(Communications Monitor)
HDRSACUE	(Store Administrator)

Note: If additional user-written control routines are supplied to interface with System Control, branch instructions to these routines must be appended, as additional one-word entries, to the System Control Second Branch Table (HDRSCBR2). These additional entries must be set up in the Master routine (see *System Control requirements*, Chapter 4, page 10).

2 Each routine, when entered, must access the TAB in order to

(a) Locate and interpret the request code and parameters currently held in the TAB

(b) Set the error reply word (word 8 of the TAB)

where necessary, issuing an error code which can be interpreted by the Error Recovery Bead.

(c) Access the Message Area (Communications Monitor) or Input/Output area (Peripheral Monitor)

The TAB and the appropriate associated area are accessed using the same method normally employed by PLAN beads, i.e. using the link addresses in HDRTABSTORE and word 22 or 23 of the TAB.

3 Before return to System Control, the service TAB HDRSCST must normally be set to indicate successful or unsuccessful processing, thus enabling Bead Scheduler to initiate error recovery when it is subsequently entered from System Control. However, if processing has been usuccessful, control may optionally be passed direct to the Error Recovery Bead, i.e. without re-entry to System Control.

Provided that the user's routines conform to the above standards, he may code them as he wishes. However, if later enhancement to multithreading is likely, the overall design of these routines should be as recommended in the relevent sections in the remainder of this Chapter.

PERIPHERAL MONITOR AND ASSOCIATED FILE HANDLING ROUTINES

Entry point/cue name

HDRPMCUE

Description

Peripheral Monitor may be written as a single PLAN segment which initiates all peripheral transfers required by the program by issuing the appropriate housekeeping macro or executing a PERI instruction each time a peripheral request is received from a bead. Alternatively, file handling may be carried out by separate routines, Peripheral Monitor being confined to controlling entry to these routines and setting the TAB and Service TAB for return to System Control.

Functions

For each peripheral request issued by a bead, Peripheral Monitor and its associated file handling routines (if any) must perform the following functions, in the order given.

1 Using parameters set up in the Request Area of the TAB by the bead issuing the current request, either:

(a) Set up any necessary control information and initiate the required transfer by means of the appropriate housekeeping macro.

or

(b) Set up a control area and initiate the transfer at PERI level.

2 If the transfer is successful, set word 8 of the TAB to zero. Otherwise place a code indicating the type of error in word 8 of the TAB (a list of recommended error codes is given in Appendix 2). The error recovery bead may, if required, be CALLed directly at this point and further processing in respect of the current request abandoned.

3 Set the service TAB HDRSCST as follows:

If the transfer has been successful:

Word 0 2

Word 1 0

If the transfer has been unsuccessful:

Word 0 3

Word 1 0

4 Branch to System Control using the instruction

BRN HDRSCCUE

Recommended routine

SYSTEM CONSIDERATIONS

Many users of single threading systems will initially prefer to write Peripheral Monitor as a single PLAN segment, handling all requests for all files by issuing the appropriate housekeeping macro in each case. In systems where only a small number of files are to be accessed, this approach has the advantage that the routine will be very

easy to write. However, where access is required to a large number of files, and in particular if later enhancement to multithreading is likely, it is recommended that file handling is carried out by separate routines coded at housekeeping and/or PERI level, as appropriate.

This latter approach has the following main advantages:

1 The program retains its modular characteristics, allowing for easy enhancement of the program's file handling capability.

2 The file handling routines will require little if any modification in order to operate in a multithreading environment.
Note, however, that if the file handling routines operate by issuing housekeeping macros, only limited multithreading will be possible due to the characteristics of the housekeeping packages concerned (see Chapter 1, page 11). It is therefore normally recommended that file handling by the routines is done at PERI level.

Coding

PERIPHERAL MONITOR

If Peripheral Monitor is written as recommended in the previous section, the routine may be limited to carrying out three fairly simple actions, namely

1 Determine the file required and branch to the appropriate file handling routine.

2 On return from the file handling routine, examine the routine's reply word and set word 8 of the TAB and

word 0 of the service TAB to indicate successful or unsuccessful processing.

3 Branch to the required file handling routine.

The user is recommended to set up a table of branch instructions to the file handling routines, either within Peripheral Monitor or within the Master routine. Peripheral Monitor may then be coded so that, for each request, it locates and executes the appropriate branch instruction in the table by means of a modified OBEY instruction.

The value used as a modifier will be obtained or calculated from parameters set up in the Request Area of the TAB by the bead issuing the current request. The format of the TAB by the bead issuing the current request. The format of these parameters, and their location in the Request Area, is largely decided by the user (see *Peripheral requests*, Chapter 3, page 11). However, if one file handling routine is used per direct access file or magnetic tape device/file, the user is recommended to allocate a *file reference number* to each file (and hence, implicitly, to each file handling routine), the files being numbered in ascending order, starting at 0. If the branch instructions to the file handling routines are stored in the appropriate sequence, a bead issuing a peripheral request will be able to specify the file required and provide Peripheral Monitor with the correct modifier by simply inserting the file reference number of the file to be accessed into a suitable location within the request area of the TAB.

Example

```
#LOWER          COMMON/FHBRANCH

          BRN   FHR0           [ACCESS FILE 0

          BRN   FHR1           [ACCESS FILE 1

          BRN   FHR2           [ACCESS FILE 2

          BRN   FHR3           [ACCESS FILE 3

            .                    .
            .                    .
            .                    .
            .                    .
            .                    .
            .                    .
          BRN   FHRn           [ACCESS FILE n
```

Check for input/output error

On re-entry to Peripheral Monitor from a file handling routine, Peripheral Monitor must check the error reply indicator HDRFHRER1. The contents of HDRFHRER1 must be transferred to word 8 of the TAB and the Service TAB HDRSCST then set as follows:

TAB setting	*Service TAB setting*
Word 8 = 0000	Word 0 = 2 Word 1 = 0
Word 8 = user's error code (i.e. non-zero)	Word 0 = 3 Word 1 = 0

Branch to System Control

Peripheral Monitor branches to System Control on the instruction:

BRN HDRSCCUE

FILE HANDLING ROUTINES

A file handling routine is normally written on the following assumptions:

1 Peripheral transfers are to be initiated at PERI level.

2 For each transfer, all the information required to set up the control area is to be obtained from parameters placed in the Request area of the TAB by the bead issuing the current request.

The routine should perform the following functions, in the order given.

1 Set up a control area, using the relevant parameters in the TAB.

2 Issue a PERI instruction.

3 Set the error reply indicator according to whether the transfer has been successful or unsuccessful. This is done by placing the appropriate parameter in the location HDRFHRERI.

4 Branch to the re-entry point in Peripheral Monitor.

The only mandatory standard to be observed is that HDRFHREI

must be set to zero following a successful transfer.
A list of recommended error codes is given in Appendix 2 .

COMMUNICATIONS MONITOR

Entry point/cue name

HDRCMCUE

Description

Communications Monitor is normally written so that it interfaces with one or more of the following standard 1900 housekeeping packages

7900 Communications housekeeping

Multiplexor housekeeping

AVDU housekeeping mark 2

The routine controls the operation of all communications devices used by the program, initiates input and output of messages, and attempts to recover from any errors occurring during transmission which cannot be automatically dealt with by the housekeeping, such as line failure.
It also deallocates the TAB from a message which has been fully processed and allocates it to the next input message.

A bead issuing a communications request of the recommended format (see *Communications requests*, Chapter 3, page 13) will set up the parameters required for entry to housekeeping within the Request area of the TAB. Communications Monitor must be written so that it will place these parameters in

the user's housekeeping control area HMPUC and then execute the appropriate macro or sequence of macros. The macros required for each housekeeping function, and the required settings of HMPUC in each case, can be found by consulting the appropriate Chapter of the manual *Data Communications and Interrogation*.

Functions

For each communications request issued by a bead, the routine must execute the following functions, not necessar: in the order given.

1 Set HMPUC for entry to housekeeping

2 Initiate output and/or input of a message by means of the appropriate macro(s).

3 If processing of a message is complete, deallocate the TAB from the current (output) message and allocate it to the next input message.

4 Attempt error recovery where necessary.

5 Set word 8 of the TAB and words 0 and 1 of the service TAB to indicate successful or unsuccessful processing.

6 Branch to System Control.

A number of additonal functions may be requested using standard communications request codes and parameters (see Appendix 1). It is left to the user to decide whether his routine is to be capable of interpreting and acting on these requests.

SET HMPUC

The parameters required for the next entry to housekeeping are obtained from the TAB and stored in the appropriate locations within HMPUC. The location within the message area at which the transfer will begin is calculated by adding the displacement value in word 3 of the TAB to the message area start address in word 22. Other relevant parameters are copied across directly.

OUTPUT AND INPUT

In a single threading environment, a bead is usually limited to issuing one of the following two main types of request

1 Ouput only (reply message generated in the course of processing the current input message).

2 Output and deallocation of the TAB(processing of the current input message is complete).

In the first case Communications Monitor must issue a suitable housekeeping macro, typically MPPUT, and set the TAB and Service TAB before branching back to System Control.

Where deallocation of the TAB is specified by the bead (i.e. bit 9 of the current request code is set to 1) this implies that processing of the current message is complete, and that the program is therefore ready to accept another input message.

In this case, Communications Monitor must output the final reply message, if there is one, and either obtain the next input message from housekeeping (MPGET macro) or if necessary suspend until a message is available (MPSUS macro). Once a message has been input, the TAB must be deallocated from

the previous message and allocated to the new one. This is done by loading the input message into the TAB message area and setting word 12 of the TAB to the bead number (1) of the Initial bead.

Device tables

If the routine is to be capable of outputting messages to more than one terminal in the course of servicing an individual request, the user is recommended to set up a suitable table of device numbers either within Communicatic Monitor or within the Master routine. Provided that the beads issue communications requests of the recommended format, the parameters required by communications Monitor to progress down this list are found in words 6 and 7 of the TAB.

ERROR RECOVERY

It is recommended that errors occurring during transmission are as far as possible dealt with within Communications Monitor. For example, re-start procedures should preferably be included in this routine.

In the event of an error occurring which cannot be dealt with, control should be passed either to the Master routine (if a suitable re-entry point has been provided) or to the Error Recovery bead. In the latter case, word 8 of the TAB must be set to the appropriate error code: control may then be passed directly to the Error Recovery bead or else the service TAB set and control transformed in the conventional manner via System Control and Bead Scheduler.

TAB AND SERVICE TAB SETTINGS

Word 8 of the TAB must be set as indicated below before exit. The service TAB must also be set as shown if System Control is to be re-entered.

	TAB setting	*Service TAB setting*
Following successful processing:	Word 8 = 0000	Word 0 = 2 Word 1 = 0
Following unsuccessful processing:	Word 8 = user error code	Word 0 = 3 Word 1 = 0
Following TAB deallocation/allocation:	Word 12 = Bead number of Initial bead = 1	-

EXIT

Exit from Communications Monitor is normally by means of a branch to System Control on the instruction

BRN HDRSCCUE

If the routine is written so that it can pass control directly to the Error Recovery bead, the required instruction following detection of an error condition is

CALL 1 *beadname*

where *beadname* is the name of the Error Recovery bead.

STORE ADMINISTRATOR

Entry point/cue name

HDRSACUE

Description

The beads in a single threading program may be written so that they are capable of obtaining additional store if the program is subsequently changed to multithreading with dynamic store allocation facilities.

If any such beads are present in the user's single threading program, a dummy Store Administrator must be provided to simulate servicing of store requests issued by the beads. The only actions required of the routine are as follows:

1 Set word 8 of the TAB to zero (0000)

2 Set the service TAB HDRSCST as follows:

Word 0 2

Word 1 0

3 Branch to System Control on the instruction

BRN HDRSCCUE

Chapter 6

The multithreading Driver

INTRODUCTION

A complete multithreading Driver may be assembled using the standard multithreading routines described in this Chapter in conjunction with User-written file handling routines. Most of the standard routines are not suitable for replacement by user written routines and are therefore not described in detail in this manual. This chapter is mainly concerned with the other methods available to the user in adapting Driver to meet the specific requirements of his program. These methods can be summarised as follows:

1 Choosing from alternative versions of the same standard routine. The choice of routines for inclusion in the user's Driver is governed by the following considerations.

(a) Whether overlaying of beads is required (alternative versions of Request Analyser and Bead Scheduler)

(b) The type of communications housekeeping in use (alternative versions of the *interface* and *queue-selector* routines within Communications Monitor)

(c) Whether first-in first-out or *bead priority* queueing of TABs is required within Bead Scheduler (alternative versions of Bead Scheduler).

2 Omitting standard routines not required by the program. Use of the following routines is optional

(a) Store Administrator is only required by programs supporting a dynamic store system

(b) Overlay Control and %EROL are only required by programs supporting an overlay system.

3 Adding user coding at Standard User entry points

A number of one-word common blocks known as user entry points are set by the user when writing the Master routine. By placing suitable CALL instructions in these locations, the user can cause his own subroutines to be entered at various points in the execution of Communications Monitor, Peripheral Monitor and Bead Scheduler. The precise nature of these subroutines is in all cases entirely at the discretion of the user: however, the entry points are provided to allow implementation of the following non-standard facilities.

(a) Message priority queueing of TABs

All TAB queues maintained by the standard Driver routines are first-in first-out: in nearly all cases TABs are placed on queues by means of the standard queueing subroutine HDRATQ (Add to Tail of Queue).

Two words of each TAB (words 25 and 26) are available so that the Initial bead can assign processing priorities to messages on input. The CALLs to HDRATQ at various points within Driver appear at user entry points and may hence in each case be replaced by a CALL to a user written subroutine for analysing these parameters and accordingly determining the position at which each TAB is to be inserted into its required queue. The act of actually

queueing each TAB may in turn be initiated from within the users subroutine by calling the appropriate queueing subroutine: this may be either HDRATQ, the standard queueing subroutine HDRAHQ (Add to Head of Queue) or a user written subroutine for inserting the TAB in an intermediate position within the queue. Full details are given on page 30 .

(b) Manipulating housekeeping control and reply information within Communications Monitor

Three user entry points within Communications Monitor enable user subroutines to be called immediately after input and before and after output.

4 Setting Driver constants to the appropriate values.

By specifying suitable constants in standard locations when writing the Master routine, the user can:

(a) Allow for operator intervention (for example closedown or major error recovery procedures).

(b) Specify the frequency at which full scans are to occur.

(c) Control the operation of the communications housekeeping by providing the usual user-variable parameters.

THE STANDARD ROUTINES

Driver structure

System Control, Store Administrator and the alternative versions of Request Analyser and Bead Scheduler each consist of a single module, as in the single threading Driver. However, both Communications Monitor and Peripheral Monitor consist of a hierarchy of modules: the modules within each hierarchy interact to perform the complex scheduling and re-cycling functions required of these routines.

Each entry to Driver on a request from a bead involves the execution of a suitable sequence of the main modules. These consist of the modules provided as Request Analyser, System Control, Store Administrator and Bead Scheduler, together with the Communications Monitor Entry/Exit routine and the Peripheral Monitor Entry routine. The latter two routines each schedule and initiate the servicing of requests via the appropriate sequence of dependent modules in their repective hierarchies.

Functions that may be or are required at many different points in the execution of Driver are performed by common subroutines. Each main module and certain dependent modules are allowed entry to a suitable selection of these subroutines.

The table below illustrates the hierarchical relationship between standard routines currently available.

Driver routine	*Main module*	*Dependent modules*	*Common subroutines*
Request Analyser	Request Analyser 2 (non-overlay version, or Request Analyser 3 (overlay version)	None	Queueing subroutine 1 Enter Exception Mode Overlay Control (version 3 only)
System Control	System Control 2	None	None
Peripheral Monitor	P.M Entry routine	P.M Queue-selector routine P.M Cycle routine File handling routines (user-written) P.M Continuation routine Exit routine	Queueing subroutines 1,2, and 3 Deallocate TAB Enter Exception Mode Return to Normal Mode

Driver routine	*Main module*	*Dependent modules*	*Common subroutines*
Communications Monitor	C.M Entry/Exit routine	C.M Queue-selector 1 (7900 housekeeping) or C.M Queue-selector 2 (Multiplexor housekeeping) *7900 or multiplexor interface routine, comprising the following*: Entry routine Test for exceptions routine Manipulate configuration routine Close routine	As Peripheral Monitor

Driver routine	*Main module*	*Dependent modules*	*Common subroutines*
		1/0 cycle routine PUT routine GET routine Final loop and exit routine	
Store Administrator	Store Administralor	None	Queueing subroutines 1,2, and 3
Bead Scheduler 2	Bead Scheduler 2 (non-overlay, first-in first-out queueing) or Bead Scheduler 3 (overlay, first-in first-out queueing) or	General Suspension Subroutine 1 (direct response mode) or General Suspension Subroutine 2 (normal suspension mode)	Queueing subroutines 1,2, and 3. Enter Exception Mode Overlay Control (versions 3 and 5 only).

Driver routine	*Main module*	*Dependent modules*	*Common subroutines*
	Bead Scheduler 4 (non-overlay, bead priority queueing) or Bead Scheduler 5 (overlay, bead priority queueing)		

Outline of operation

GENERAL

Normal mode and Exception mode.

Driver runs in normal mode so long as no errors occur in the servicing of request. However, when such an error occurs, a standard common subroutine, known as the Enter Exception mode routine, is entered and initiates a phased shutdown of all message threads other than the one in which the error occurred. Once this closedown has been completed, the program is in exception mode. It continues single-threading operation until the error is cleared, whereupon the Return to Normal mode subroutine is called and the program is brought back full multithreading operation.

Full scans

As in the single threading Driver, entry to the Multithreading Driver is always as a result of a specific request for a facility such as output to a particular peripheral device, transfer of control to a particular bead, and so forth. However, each time that a user-specified number of entries to Driver have been made, a full scan will occur. In this case, entry will be made to all the main Driver routines and not just these specified in the current request.

Full scans are necessary because each Driver routine, whilst servicing a request, may accumulate further requests for the same facility. Full scans enable such requests to be dealt with without waiting for a specific request for the individual routine.

REQUEST ANALYSER

Two version of this routine are available to multithreading users: one for non-overlaid programs (Request Analyser 2) and one for overlaid programs (Request Analyser 3).

As in the single threading Request Analyser, both versions of the multithreading Request Analyser validate each request passed to Driver, store the contents of accumulators 0 and 4 to 7 in the TAB concerned, and set the Service TAB for entry to System Control. They also perform the following three functions.

1 When a bead is found to have completed its execution (that is, word 10 of the relevant TAB is set zero) Request Analyser updates the relevant free/busy indicator in the table HPRDT3. This indicator is subsequently examined by Bead Scheduler, and informs the routine that control may now be passed to the bead in respect of another TAB.

2 Request Analyser also initiates full scans within Driver at user-specified intervals by reducing the value of word 0 of the Driver count HDRDCT by 1 on each entry and setting the service TAB for a full scan whenever the count reaches zero. The count is then reset to its original value as specified in word 1 of HDRDT.

3 Before exit to System Control, the TAB is placed on the relevant queue awaiting servicing of its request. The subsequent progressof TAB through the various Driver queues is described in the section *The queueing system*, page 21.

Request Analysor 3 has the additional function of initing overlay of beads whenever possible. This function is described in the section *The overlay system*, page 39.

SYSTEM CONTROL

The only difference between the single and multithreading versions of System Control is that the latter uses an additional branch table (HDRSCBR1) instead of the second branch table (HDRSCBR2) during a full scan.
By OBEYing each of the CALL instructions in HDRSCBR1 in sequence, System Control enters each of the main modules in turn in a sequence decided by the user (see Chapter 4, page).

PERIPHERAL MONITOR

An outline flowchart of Peripheral Monitor is shown in figure . The functions of the standard routines within Peripheral Monitor are described below: the common subroutines and user-written file handling routines are discussed seperately in the appropriate sections of this chapter.

Entry routine

Peripheral Monitor is always entered via the P.M. Entry routine. This routine validates each request, tests the condition of the program (normal mode or exception mode), and the type of entry (full scan, specific peripheral request, or both). It then passes control to the appropriate dependent routine or common subroutine, if necessary first calling the queue-selector subroutine to queue the TAB containing the incoming request and provide parameters for entry to the appropriate file handling routine.

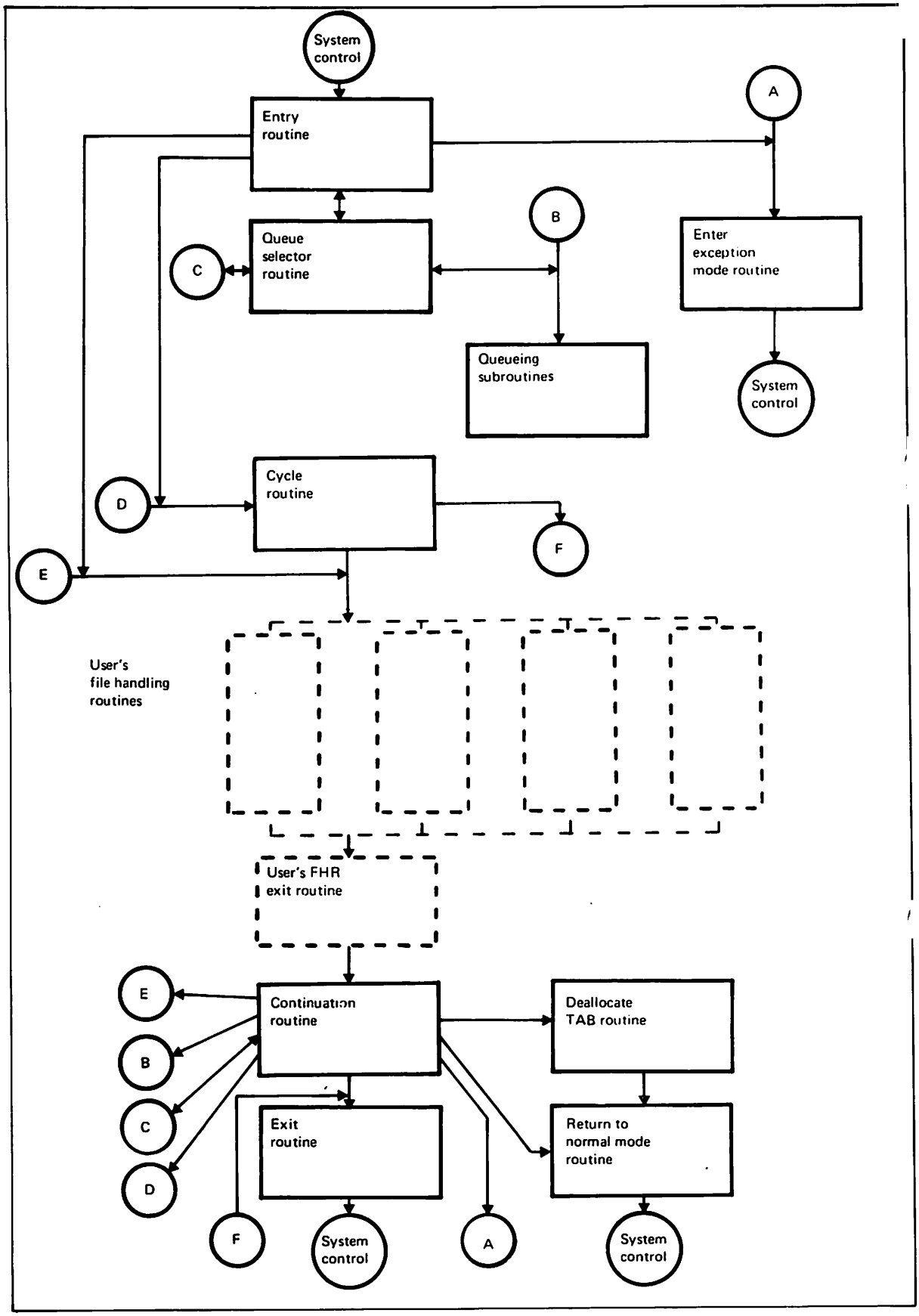

Figure 1 Peripheral Monitor - outline flowchart

Cycle routine

This routine is only entered in the course of a full scan. It causes control to be passed to each file handling routine not currently engaged in a peripheral transfer.
Re-entry after execution of each file handling routine is via the Continuation routine: after initiating as many transfers as possible the cycle routine passes control to the Exit routine.

Continuation routine

This routine determines the subsequent action to be taken whenever a file handling routine completes its execution. This includes removing TABs, whose requests have been serviced from their queues (direct entry to the standard queueing subroutines) moving TABs from one queue to another (entry to Queue-selector subroutine) or making entry in the appropriate cases to one of the standard common subroutines.

Exit routine

This routine sets the Service TAB and exits to System Control.

COMMUNICATIONS MONITOR

An outline flowchart of Communications Monitor incorporating the 7900 interface routine is shown in figure 2 . The function of the standard routines are as follows.

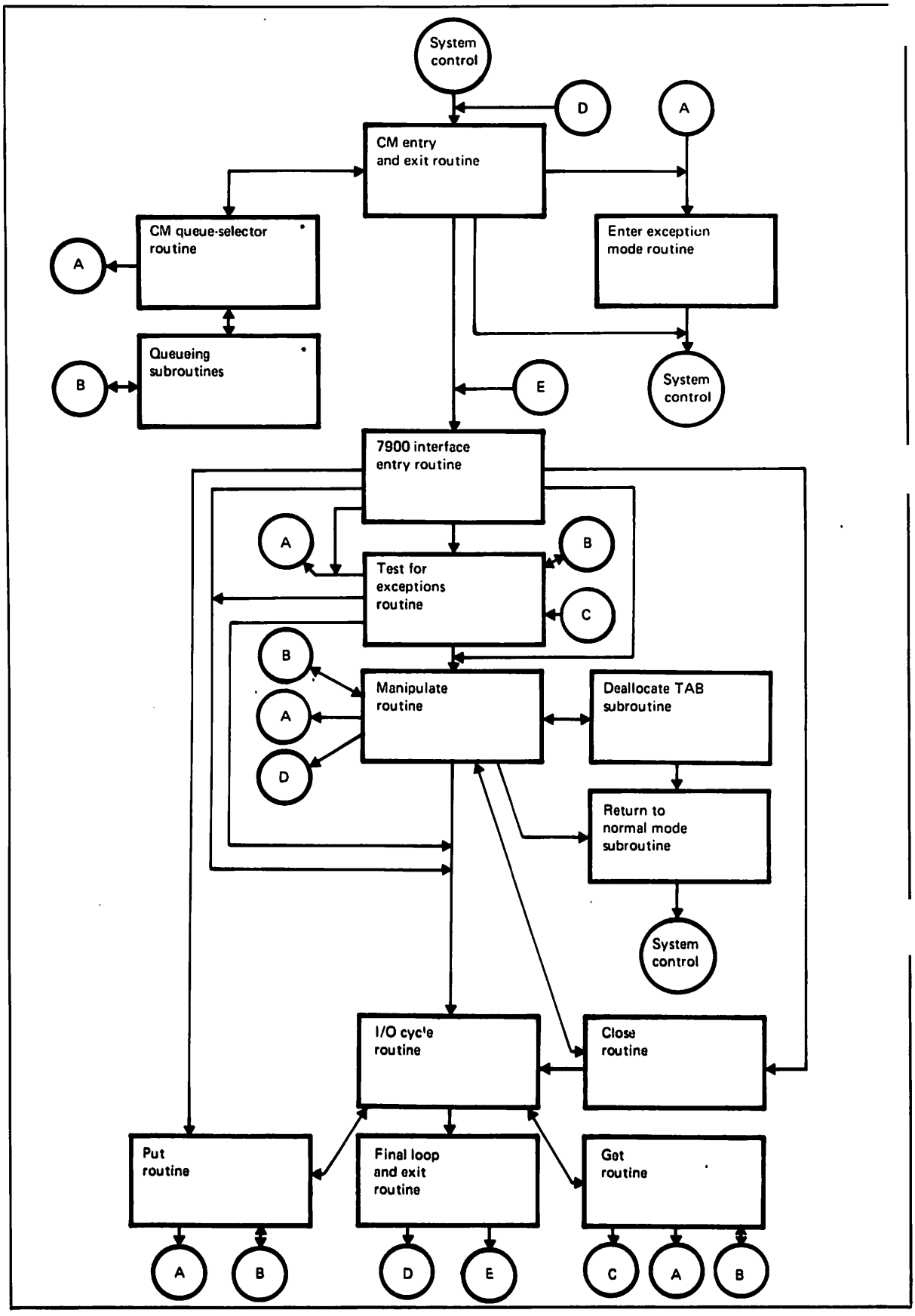

Figure 2 Communications Monitor - outline flowchart

C.M. Entry/Exit routine

This routine ch .cks incoming requests in the same manner as the P.M. entry routine. However, provided that the request is valid, it calls the queue-selector sub routine to queue the incoming requests, and then passes control to the appropriate interface routine. When processing by the interface routine is complete, the Entry/Exit routine also sets the service TAB and passes control back to System Control.

7900 interface routine

The functions of the various modules within this routine is described below. The multiplier version is identical except that it maintains only one queue each for output requests and requests referred because of overload.

1 The entry routine checks whether the program is in exception mode and if so passes control to the routine which will handle the current request. If closedown of the communications network is required, it passes control the the Close routine. If requests for communications output have been previously rejected because of overload, the TABs concerned are restored to the appropriate queues (see *Communications Monitor queues*, page 24) and control passed to the Test for Exception routine.

2 The Test for Exception routine checks whether there are any TABs available for processing of error conditions. If this is not the case, it passes control the input/output cycle routine. It also tests for operator intervention by performing a logical AND on the contents of word 30 and the users constant HDRCM01: if an operator message has been recived, a TAB is allocated to that message and control passed to the Enter Exception Mode common subroutine. Otherwise, the routine proceeds to

test the state of the communications network using the indicator HMPIF and macro MPTES. If an error condition is reported, it will set up a TAB containing the appropriate error code and pass control to the Enter Exception Mode subroutine.

3 The Manipulate routine services requests to open, or change the state, the teleprocessors in use. It also services requests for deallocation of TABs and return to normal mode by passing control to the appropriate common subroutine. In appropriate circumstances, it enters the Close routine to initiate closedown of the communications network.

4 The Close routine sets indicators to inhibit input and output, and then closes down the Communications network.

5 If output requests are waiting to be serviced, and the buffers are free, the Input/Output Cycle routine calls the PUT routine to initiate output. Subsequently, if a message is found to be awaiting input and a free TAB is available for allocation, the GET routine is called to obtain input.

6 The PUT routine initiates output by means of an MPPUT macro. According to the reply information received, the TAB concerned is then either:

(a) Queued to await servicing by Bead Scheduler, or, if deallocation of the TAB is required, passed to the Deallocate TAB Common Subroutine.

(b) If overload has occurred, removed to another queue awaiting the next entry to Communications Monitor.

If the reply to the MPPUT macro indicates that an error has occurred, the Enter Exception Mode common subroutin

is entered.

7 The GET routine initiates input by means of an MPGET macro. If an error occurs in input, the Enter Exception Mode common subroutine is entered. Otherwise a TAB is allocated to the incoming message and queued awaiting servicing by Bead Scheduler.

8 The final loop and exit routine checks that as many requests as possible have been serviced on the current entry. It then passes control back to the final loop and exit routine.

STORE ADMINISTRATOR

Store Administrator allocates cells of store from the users common pool to TABs. A cell is allocated as the input/output area of a TAB by Store Administrator placing the start address of the cell in word 23 of that TAB. Similarly, a cell allocated as an Additional Core area has its start address stored in word 24 of the TAB requiring it. The methods employed by Store Administrator in allocating cells of various sizes to TABs are described in the section *Store Administrator queues*, page 25 .

BEAD SCHEDULER

Alternative versions of Bead Scheduler are available offering a choice of TAB queueing systems (Bead priority or first-in first-out) and allowing for implementation of bead overlay if required. These features are described in the section *The queueing system* and *The overlay system* elsewhere in this chapter. All versions are otherwise basically similar in their operation to the single-threading Bead Scheduler.

SUBROUTINES

Queueing subroutines

Three standard queueing subroutines are provided and are required in both standard and non-standard queueing systems. They are:

1 HDRATQ (Add to tail of queue)

2 HDRRFQ (Remove from head of or intermediate position in queue)

3 HDRAHQ (Add to head of queue)

The subroutines are used to perform queueing of TABs and store cells.

Deallocate TAB

This subroutine is entered at the end of processing a message and clears the TAB concerned of all parameters relating to the completed transaction. In a system employing dynamic store allocation, it also checks that store cells have been returned to the common pool and performs this function if required.

Enter Exception Mode

This subroutine will be entered when any Exception condition is detected by Driver and will place the system in single thread working, and set parameters that will cause System Control to enter Bead Scheduler. Bead Scheduler in turn will enter the user's Error Recovery Bead. If the system is already in Exception

Mode a user entry point will be OBEYED.If return is made from this point the Program will be halted.

Return to Normal Mode

This routine is entered to restore the multi-threading operation of the system when an exception condition has been cleared. It may be entered in response to a direct request from a bead (typically the Error Recovery bead).

General Suspension subroutine

This routine will be entered from Bead Scheduler whenever the latter routine is unable to activiate a bead. It will perform the following functions.

1 If a full Driver Scan has not just been implemented it will cause one to take place in order to ensure that there are no outstanding requests awaiting servicing by any of the Driver routines.

2 If a full Driver Scan has just been implemented it will either suspend Member 0, or halt the program. The latter course will be followed if both of the following conditions are true.

(a) Indicators have been set showing that all communications Interface routines have been finally closed.

(b) All tabs are on the free TAB queue (see page)

Two versions of the routine are available, incorporating two distinct methods of suspending Member 0, Version 1 is for systems in which either the combination of Trusted facility and Real Time Clock makes possible a simple suspension on a SUSIN instruction or all peripheral are in Direct Response mode, Version 2 is for systems in which neither of these circumstances occur.

THE QUEUEING SYSTEM

Communications Monitor, Peripheral Monitor, Store Administrater and Bead Scheduler all maintain queues of TABs. When a bead issues a request for a particular Driver facility, the routine to which the request is passed does not usually service it immediately, but places the TAB concerned on a queue of TABs continuing previously issued requests for the same facility. Under the Standard Driver queueing system, queued requests are usually serviced in order of arrival: this is hence known as a first-in first out system.

The organisation of these queues, and the role of each in the operation of Driver, is described below. The information given is principally for the benefit of users who may wish to modify the standard system first-in first-out system in order to implement their own system of processing by message priority.

Queue organisation

All TAB queues are organised in the manner shown in figure 3. . The user specifies a 2-word *queue management area* (QMA) for each queue when writing the Master routine.

On the initial entry to Driver, all TABs other than the one used for entry will be held on the Communications Monitor free TAB queue awaiting allocation to input messages. Accordingly, word 0 of the user's QMA for this queue is set in the Master routine to the start address of the first TAB to be allocated to a message: this TAB is said to be at the *head* of the queue. Similarly, word 1 of the QMA is set to the start address of the TAB that will appear at the *tail* of the queue on entry to Driver. The actual queue is formed by presetting word 21 of each TAB to the start address of the next TAB in the queue. Word 21 of the TAB

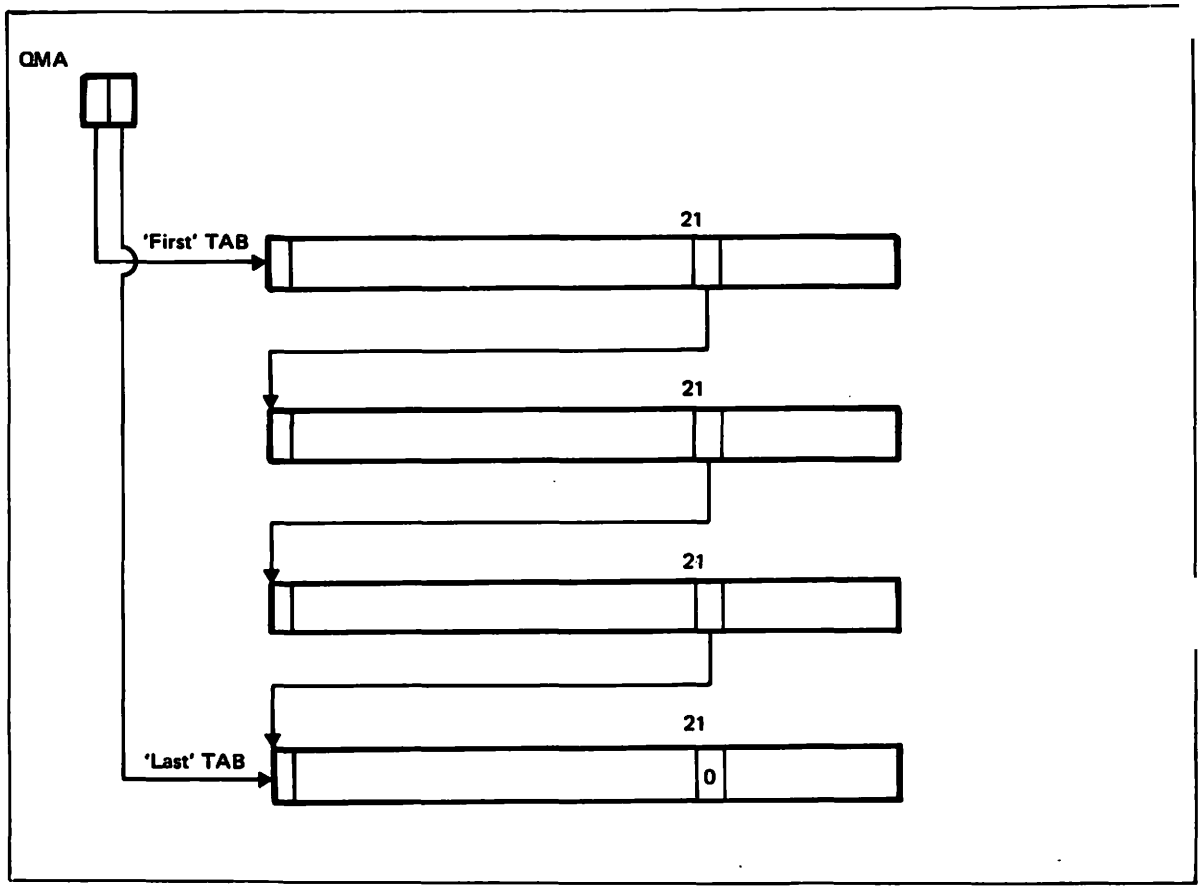

Figure 3 Organization of a TAB queue

at the tail of the queue is set zero.

All other TAB queues will be empty in the initial entry to Driver, and their QMAs are accordingly set zero in the Master routine. TABs begin to accumulate on these queues as soon as the beads begin issuing requests. Driver forms queues and moves TABs from one queue to another by updating the relevant TAB addresses in the QMAs and word 21 of each queued TAB. The manner in which this is done is described below.

Queue manipulation

ADDING A TAB TO A QUEUE

Except when carrying out certain specialised functions not accessible to the user, all Driver routines using the standard first-in first-out system will queue TABs passed to them for servicing by placing each TAB at the tail of the appropriate queue.

A TAB is placed at the tail of a queue by passing its start address and the start address of the appropriate QMA as parameters in a call to the subroutine HDRATQ. The procedure is then as follows:

1 If the queue is currently empty, (words 0 and 1 of QMA = 0) the incoming TAB will form both the head and tail of the queue. Accordingly, words 0 and 1 of the QMA are both set to the start address of the TAB. Word 21 of the TAB is set zero.

2 If one or more TABs are already present on the queue, the TAB at the tail of the queue is located by means of the address in word 1 of the QMA. Word 21 of this

TAB is set to the start address of the incoming TAB. Word 21 of the incoming TAB is set zero and its start address stored in word 1 of the QMA.

The user wishing to implement a message priority queuing system can substitute calls to his own subroutines for calls to HDRATQ at the various user entry points within Driver described in Chapter 4. A recommended approach to writing the requisite subroutines is given in the section *Message priority queueing*, page 30 .

REMOVING A TAB FROM A QUEUE

Under all queueing systems, the TAB currently appearing at the head of a given queue will have its request serviced before any other TABs in the same queue. The TAB remains at the head of the queue whilst its request is serviced. The Driver routine then calls the standard queueing subroutine HDRRFQ: this subroutine removes the TAB from the head of the queue by the following procedure.

1 The start of the next TAB in the queue is found in word 21 of the outgoing TAB and copied into word 0 of the QMA. If there are no further TABs in the queue, words 0 and 1 of the QMA are set zero.

2 Word 21 of the outgoing TAB is set zero.

The TAB has now been removed from its queue, and HDRATQ may be called if necessary to place it at the tail of another Driver queue.

Operation of the Standard queueing system

PERIPHERAL MONITOR QUEUES

Peripheral Monitor usually maintains a seperate *device queue* for each file handling routine. Once a TAB has had a request serviced by a file handling routine, it is placed on one of the following queues.

1 If a request has been made in respect of several peripheral devices, and not all of these have yet been accessed, the TAB is placed on the next appropriate device queue

2 If the TAB is to be deallocated, and its request has otherwise been fully serviced, the Deallocate TAB subroutine is called to clear the TAB of all variable parameters relating to the serviced request. The TAB is then placed on the free TAB queue awaiting servicing (that is, allocation to an input message) by Communications Monitor.

3 If the TAB is to be accessed by another bead, it is placed on Bead Scheduler's *external queue* awaiting further queueing and eventual servicing by Bead Scheduler.

COMMUNICATIONS MONITOR QUEUES

All TABs containing requests for communications output are placed on an *output queue* awaiting servicing by the interface routine in use. Where a multiplexor interface routine is in use within Communications Monitor, a single output queue is maintained. The 7900 interface routine, however, maintains a seperate output queue for each teleprocessor.

In both cases, any TABs whose requests involve altering the configuration or handling of a communications device (for example, 'Change priority' or closedown requests) are placed on a separate *manipulate queue* to await servicing.

Each output queue has a corresponding *wait queue*. If the requests held in a particular output queue cannot be serviced without overloading the communications processor/multiplexor stack, the TABs are placed in their existing order in the corresponding wait queue . This feature is necessary because Communications Monitor may cycle through its queues several times in a single pass and may hence cause overload if the TABs ar not removed from their output queue at the appropriate time. The TABs will be restored to their output queue on the next entry to Communications Monitor, being placed in fron of any TABs that have accumulated on the queue in the meantime.

Deallocation of TABs, when required, is carried out by use of the Deallocate TAB subroutine, as in Peripheral Monitor. The resulting free TAB queue is located whenever an input message is received, the TAB at the head of the queue being allocated to the incoming message. This TAB is then placed on the Bead Scheduler external queue to await queuing and servicing by Bead Scheduler.

All other TABs whose communications requests have been fully serviced are also placed on the Bead Scheduler external queue.

STORE ADMINISTRATOR QUEUES

Store Administrator maintains two queues of TABs per block of store cells set up by the user. One queue, known as the *first block queue*, holds TABs which have already had one

store cell allocated to them and are now awaiting allocation of a second cell. The other queue, known as the *second block queue*, holds TABs whose requests for store have not yet been serviced at all.

Each time Store Administrator allocates store cells from a particular block, it will give priority to the TABs held in the first block queue for that block. This ensures that the servicing of partially satisfied store requests is completed as quickly as possible. TABs whose requests for store have been fully serviced are removed to the Bead Scheduler external queue.

One as many TABs as possible in the first TAB queue have had store allocated to them, Store Administrator attempts to service the requests of the TABs in the second block queue. Once a TAB in this queue has had store allocated to it, either of the following actions will occur.

1 If the TAB required only one store cell, its servicing is complete and it is placed on the Bead Scheduler external queue.

2 If the TAB requires a further store cell, it is placed on the first block queue for the block containing store cells of the appropriate size. For example, suppose a TAB has just been allocated a 128 word Input/Output area and now requires a 512 word Additional Core area. In this case, Store Administrator will remove the TAB from the second block queue for the block of 128 word cells, and will place it on the first block queue for the block of 512 word cells.

It may be noticed in passing that Store Administrator handles each block of store cells in the common pool as a first-in first-out queue. The blocks are set up in the Master routine as chained queues of cells: whenever Store Administrator

allocates a call to a TAB, it does so by removing a cell of the required size from the head of the appropriate queue and placing the Start address of the cell in word 23 or 24 of the TAB, as required. Similarly, on deallocation, each cell is returned to the tail of the appropriate queue. Queue manipulation is performed throughout by the standard queueing subroutines, using QMAs of standard format.

THE BEAD SCHEDULER EXTERNAL QUEUE

The servicing of a bead's request can terminate in two ways:

1 By deallocation of the TAB concerned following servicing by Peripheral Monitor or Communications Monitor. This only occurs following the find entry to Driver at the end of a message thread.

2 By entry to a bead: either by entry to the next bead in the thread, or re-entry to the bead issuing the request, or, following detection of an error, entry to the Error Recovery bead.

Thus, in the majority of cases, a TAB whose peripheral, communications or store request has been serviced by the appropriate routine requires further servicing by Bead Scheduler. When, in such cases, Peripheral Monitor, Communications Monitor or Store Administrator completes the servicing of a request, it places the TAB concerned on the Bead Scheduler's *external queue*. If Driver is entered as a result of a specific request for a bead, a preliminary entry is made to Bead Scheduler in order to place the relevant TAB on the same queue. The external queue hence contains all TABs which have been passed to Bead Scheduler for servicing since the last complete execution of that routine.

BEAD SCHEDULER INTERNAL QUEUES

Each time Bead Scheduler is entered to pass control to a bead (that is, during a full scan or following the preliminary entry described above) it transfers the TAB currently held on the external queue to one or more internal queues, depending on the type of queueing systems in use.

The alternative queueing systems available within Bead Scheduler

are described below. Note that all versions of the routine service requests for the Error Recovery bead immediately.

First-in first-out queueing (Bead Scheduler versions 2 and 4)

A single internal queue is maintained by these versions. On each entry to Bead Scheduler, the TABs are transferred, in the order that they were queued, to the tail of the internal queue.

Starting with the TAB at the head of the internal queue, Bead Scheduler determines the bead requirements of each TAB in turn. As soon as a TAB is found which requires either re-entry to a bead or entry to a bead not currently in use, the TAB is removed from the internal queue and control passed to the requested bead. The remaining TABs remain on the internal queue awaiting the next entry to Bead Scheduler.

Bead priority queueing (Bead Scheduler versions 3 and 5)

In a bead priority system, a separate first-in internal queue is maintained for each bead in the program. On each entry to Bead Scheduler, each TAB on the external queue is individually transferred to the appropriate queue. The position in which a TAB is placed on an internal queue depends on the setting of word 10 of that TAB. If word 10 is non-zero (that is, re-entry to a bead is required), the TAB is placed at the head of the queue. If word 10 is zero and thus containing a request for first entry to a bead, the TAB is placed at the tail of the queue. In each queue, therefore, requests for re-entry are given priority over requests for 'first-time' entry.

Bead Scheduler determines whether any TABs are available for entry or re-entry by scanning down the free/busy indicator

table HDRBT3. As soon as a bead is found to be free, and provided that one or more TABs are present on the corresponding queue, the TAB holding that queue is removed and Bead Scheduler exits to the bead. Since the entries in HDRBT3 are both held and scanned in ascending order of bead number, it follows that low-numbered beads automatically take precedence over higher numbered beads.

Message priority queueing

It can be seen from the foregoing description that the standard routines offer the user two choices of queueing system

1 First-in first-out queueing throughout.

2 First-in first-out queueing with bead priority queueing in Bead Scheduler.

As a further option, the user may implement a message priority system allowing for especially fast processing of selected message-types. A suggested procedure for doing this is described below. Since the factor involved in choosing between the three queueing systems are somewhat complex and vary between different applications, no general guidance can be given as to which should be selected.

ASSIGNING MESSAGE PRIORITIES

The initial bead is coded so that, on input of a message, it assigns a permanent priority and a temporary priority to the message by placing the same priority number in both word 25 and word 26 of the TAB allocated to the message. Priority numbers are assigned in ascending order of message priority: in a typical case the message with the lowest priority would

be assigned priority 0 and that with the highest priority assigned priority 5. The criteria used to determine the priority of each message are of course decided by the user.

QUEUE MANIPULATION

General

The user must write a subroutine to initiate queueing of TABs according to their relative message priorities. This subroutine is called instead of the standard queueing subrout HDRATQ at some or all of the appropriate user entry points; the chosen entry points are set in the Master routine to contain a call to the user's subroutine instead of a call to HDRATQ.

The subroutine works on the following two basic principles

1 The position at which an incoming TAB is to be placed on a TAB queue is determined by comparing its permanent priority (in word 25 of the TAB) with the temporary priority in word 26 of each TAB currently held on the queue. When a TAB is encountered whose temporary priority is lower than the incoming TABs permanent priority, the incoming TAB is inserted immediately in front of it.

2 The temporary priority of each TAB currently held on the queue is increased by 1 immediately before each of the above comparisons is made. Thus the longer a TAB remains on the queue, the less is the chance of an incoming TAB taking precedence over it. This prevents messages with a nominally low priority from being permanently locked out of the program.

Message priority queueing may be initiated in Peripheral Monitor, Communications Monitor and Bead Scheduler 4 by a single common subroutine. An additional subroutine will be required if Bead Scheduler 2 is in use-and priority queueing is required within this routine. Message priority queueing may not be implemented within Bead Scheduler 3 or 5 (bead priority versions).

Procedure

The parameters passed to the users subroutine are those intended for HDRATQ. The parameters are held in the following accumulators

X0	Link accumulator
X2	Start address of incoming TAB
X3	Start address of relevant QMA
X4	Zero

The contents of these accumulators must be restored if the user's subroutine subsequently calls HDRATQ or any of the standard queueing subroutines.

On entry to the user's subroutine, the temporary priority in word 26 of the incoming TAB must be set equal to the permanant priority in word 25 of that TAB. The action taken will then depend on the state of the QMA. The recommended procedures for the various possible settings are as follows:

1 Queue empty (QMA word 0 = QMA word 1 = 0)

In this case, no action need to be taken other than calling the standard queueing subroutine HDRATQ on the instruction

```
CALL 0 HDRATQ
```

2 One or more TABs present on the queue (QMA word 0 = QMA word 1 = 0 , or QMA word 0 ≠ QMA word 1)

The start address of the first TAB is found in word 0 of the QMA and the temporary priority in word 26 of the TAB increased by 1. The start address of the incoming TAB is found in accumulator 2.

If the permanent priority of the incoming TAB is greater than the temporary priority of the TAB heading the queue, the subroutine HDRAHQ is entered on the instruction

```
CALL 0 HDRAHQ
```

and places the incoming TAB at the head of the queue.

Otherwise, word 21 of the TAB at the head of the queue is checked. If it is zero, indicating tail-of-queue, HDRATQ is called to place the incoming TAB on the queue. If it is not zero, the start address of the queued TAB is stored in a user-reserved location: the address in word 21 of this TAB is then used to locate the next TAB in the queue. The procedure below is then followed for each queued TAB in turn until the point at which the incoming TAB is to be inserted is found.

a) The temporary priority of the queued TAB is increased by 1 and compared with the permanent priority of the incoming TAB.

b) If the permanent priority of the incoming TAB is less than or equal to the temporary priority of the TAB word 21 of thw queued TAB is examined. If this word is zero, indicating tail-of-queue, HDRATQ is called to queue the incoming TAB. If word 21 is not zero, the user's location containing the start address of the last queued TAB examined is overwritten with the

start address of the queued TAB currently under examination. The address in word 21 of the latter is then used to locate the next TAB down the queue and the procedure is repeated from a) above.

c) If the permanent priority of the incoming TAB is greater than the temporary priority of the queued TAB under examination, word 21 of the incoming TAB is set to contain the start address of the queued TAB. The start address of the queued TAB which was previously examined is then found in the user's reserved location: word 21 of this TAB is set to the start address of the incoming TAB, which is thereby inserted into the queue.

The remaining TABs in the queue are then scanned using the address in word 21 of each TAB to proceed from one to another. On finding word 21 of a TAB set zero, the user's subroutine exists on the chosen link accumulator.

An additional procedure is required if the user's subroutine is called from Bead Scheduler 2 (user entry point HDRBSUE). In this case, the user's subroutine is required to transfer all the TABs currently held on the Bead Scheduler external queue to their appropriate priority positions on the internal queue. Thus instead of only having to deal with only one incoming TAB per entry, the user's subroutine will have to repeat the priority queueing procedure described above for eac each TAB in the external queue.

In this case, the start address of the first/next TAB to be removed from the external queue is found in word 0 of the queue management area, HDRBSEXTQ. The standard queueing subroutine HDRRFQ is called on the instruction

```
CALL 0 HDRRFQ
```

with accumulators 2, 3 and 4 set as follows:

X2 Start address of TAB
X3 0/HDRBSEXTQ
X4 0

On return from the routine, priority queueing of the TAB on the internal queue is performed in the normal manner, the QMA for the internal queue being HDRBSINTQ. If word 0 of HDRBSEXTQ is then found to be set zero, the user's subroutine exits on the chosen link accumulator. If it is not, it contains the start address of the next TAB to be removed from the external queue, and the entire procedure described above is repeated.

FILE HANDLING ROUTINES

In many cases, these are the only User-written routines required for the multithreading Driver. A typical file handling routine is shown in fig. 4 .

File handling should normally be performed at PERI level for the reasons given in Chapter 1. Each time a file handling routine is entered, the following parameters will be supplied in the location indicated to enable the routine to carry out a transfer.

1 The parameters required to set up the control area for the PERI instruction are held in suitable locations, as decided by the User, within the request area of the TAB and its associated Input/Output area. The file reference number used by the bead issuing the request to reference the file handling routine is held in word 1 of the TAB.

2 The queue management area of the TAB queue for the routine is held as an entry in the table HDRFHRQ. The first word of the Management Area for any given file handling routine is held in the location HDRFHRQ+(2 x *frn*) where *frn* is the file reference number of the file handling routine. When the routine is to service a request, the start address of the TAB to be serviced is found in word 0 of this QMA.

The following indicators must be used in the manner described to maintain the interface with Peripheral Monitor.

1 A free/busy indicator must be examined following entry to the ruling and set on exit. The indicator for any given file handling routine is held in the location HDRFHRFB1. The settings of this indicator have the following significance:

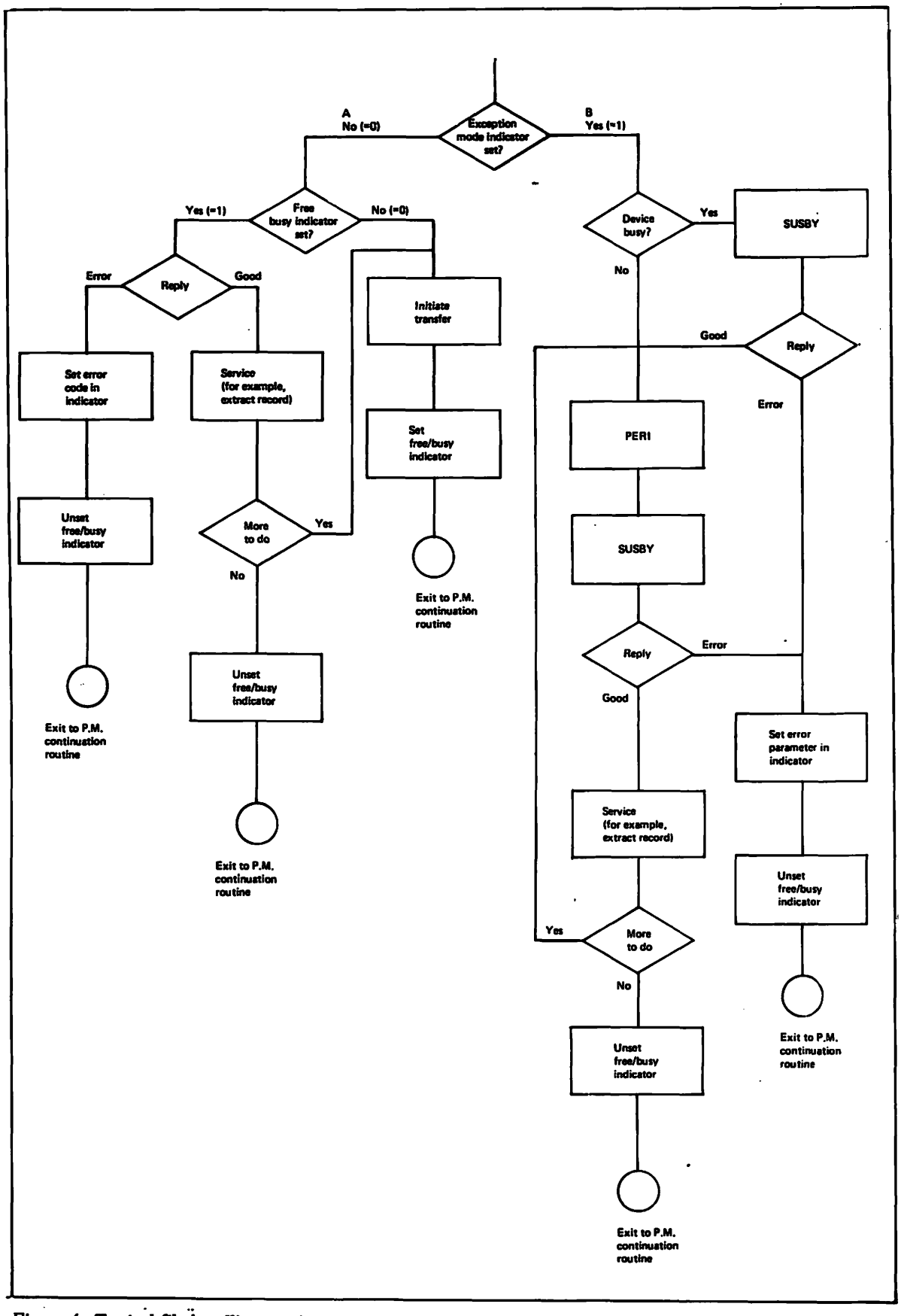

Figure 4 Typical file handling routine

(a) On entry:

HDRFHRB1	=	0	The servicing of a request was completed on the last entry and the routine is now free to perform another peripheral transfer.
HDRFHRB1	=	1	The routine initiated a transfer on the last entry to the file handling routine. A check must be made to see if the transfer is now complete.

(b) On exit:

HDRFHRB1	=	0	Servicing of a request is now complete and the routine is free to service another request on the next entry.
HDRFHRB1	=	1	A transfer is in progress and the routine is therefore not free to service any further request.

2 On completion of a peripheral transfer, and re-entry to Peripheral Monitor, Peripheral Monitor will examine the device reply word (word 1 of the control area). Peripheral Monitor locates this word by finding its address in the location HDRFHRREP + *frn* where *frn* is the file reference number of the file handling routine. If only one control area is used by the file handling routine, this address is preset and no action is required on the part of the file handling routine. However, if more than one control area is used by the file handling routine, it must set this address dynamically according to which

control area was used in the latest transfer.

3 In detection of an error, the appropriate error code must be placed in the location HDRFHERI. Peripheral Monitor will subsequently place this code in word 8 of the TAB in use before causing control to be passed to the Error Recovery bead.

Control is passed back to Peripheral Monitor by branching to the Continuation routine on the instruction:

```
BRN HDRPMCONT
```

File handling by housekeeping

A user wishing to use direct access housekeeping, but anxious to obtain a degree or multithreading, may do so by using Member 3 of his program to house a file handling routine incorporating the housekeeping.

The following limitations will apply:

1) It will only be possible to process one transaction through the housekeeping at a time (except when using physical level processing: in this context however we are dealing with users who require logical level processing).

2) Requests for several files accessed via the housekeeping will have to be placed on one TAB queue with a consequent "bottle neck" effect.

Additionally, Driver will impose the following standards.

1) Only one request at at time will be available for servicing in Member 3, namely the request held in the TAB at the head of the queue; all handling of the queue will be

controlled in Member 0,

2) The user will write a further file handling routine in addition to those held in Member 0, and when writing the Master routine will set entries in the F.H.R. tables (HDRHRFBl, HDRFHRCALL and HDDFHRREP) in the same way as for a normal file handling routine. The only exception is that the Address of the device reply word for file handling routine will in this case be the address of an indicator set negative when member 3 is activated, and returned to zero by member 3 when it has finished the processing. This will maintain the interface with the Queue Selector Cycle Routines within Peripheral Monitor. Member 3 will be able to use the negative setting of this indicator to guard against "spurious desuspension" which could otherwise lead to very unpredictable results.

The overlay system

The following routines are required to implement the Driver overlay system:

1 Request Analyser 3

2 Bead Scheduler 3 or 5

3 Overlay control

4 The standard overlay subroutine %EROL.

To allow overlay to be performed with the minimum interruption to the remainder of the program, Overlay Control and %EROL generate autonomously in member 2 of the program.

Both Request Analyser and Bead Scheduler jointly maintain

a number of indicators for each overlay bead, using the overlay tables HDROL!ND1 and HDROL!ND2. Overlay control continuously scans these indicators, initiating overlay by %EROL whenever this found to be necessary.

The main feature distinguishing the multithreading overlay system from that of the single threading Driver is that Request Analyser can in some cases anticipate Bead Schedulers requirements for overlay beads and ensure that these beads are present in main store by the time Bead Scheduler is ready to pass control to any one of them. Similarly, Bead Scheduler can initiate overlay in respect of request progress through its internal queue(s). As a result, a request for an overlay bead that is serviced within Bead Scheduler need not have its servicing delayed whilst the required bead is brought into main store. The procedure involved falls into two main steps, and can be summarised as follows.

1 Whenever Bead Scheduler transfers a TAB from its external queue, it will check each request to determine whether an overlay bead is required. Whenever this is the case, it will then check whether the required bead is already present in main store. If it is, the routine proceeds to the next TAB down the queue. If it is not, the routi will do one of the following:

(a) If, in scanning down the queue no previous (and hence higher priority) request has been found for a bead in the same overlay area, an indicator for the next bead required in that area is set to initiate overlay.

(b) If a previous request for a bead in the same overlay area has been found, an indicator for the bead is set to indicate that the bead is 'wanted' but cannot yet be overlaid. This may happen several times in each entry to Bead Scheduler: accordingly indicator

for the beads in each overlay area are set to specify the order in which the beads wanted by requests currently held on the internal queue(s) are to be overlaid.

2 Whenever Request Analyser is entered from an overlay bead, and finds that the bead has completed its execution, it checks the overlay area. If the bead that has just completed its execution is also found to be the next 'wnated' bead, no action is taken and the bead hence remains in main store. Otherwise, Request Analyser sets the appropriate inidcators for the next 'wanted' bead in the area: overlay of the bead is initiated as soon as Overlay Control scans these indicators.

Chapter 7

Program testing

1900 DRIVER TESTING AIDS

Testing Aids consist of a set of programs and routines which enables real time program beads written to 1900 Driver standards to be tested individually, or linked together. The beads may be tested without using the files or communications terminals required by the real time program.

The following Testing Aids are available:

Trial Data Set Up (#XJBA and #XJBB)
General Bead Tester (subroutine groups SDAT and SDAD)
Selective Print (#XJBC and #XJBD)

Full details of these programs and routines will be found in Chapter 8 of this manual. A general description of their use is given in Chapter 4 of the manual *Introduction to 1900 Driver*.

TESTING BEADS

A bead should initially be tested independently, using the 1900 Driver testing aids, preferably by the programmer who wrote the bead. Then, still using the testing aids, beads may be tested linked together, leading to a linked test of all the beads required in the processing thread for one message type. If required, this technique may be extended to linked testing of the complete processing element.

Careful planning of the trial data for the initial testing of individual beads should allow the same data to be used

for later linked testing. Ultimately the trial data prepared for testing of the Initial bead may be used, in the final stages of bead testing, to test the complete linked processing element.

Where enhancements to the program require the writing and testing of new beads, there beads should first be tested individually using the Driver Testing Aids. Later testing will involve linking of these beads with one another and with fully tested beads already in use is the live on-line program. Again the final stage of testing may be a test of the entire enhanced processing element.

TESTING OTHER USER WRITTEN ROUTINES

User written control routines may be tested independantly using a technique similar to that employed by General Bead Tester. A suitable user written test routine is required which will simulate the appropriate interface, read from a card reader or paper tape reader the data to be input to the routine under test, and record, on a line printer, the data output by the routine. For example, to test a Peripheral Monitor routine for a single-threading program, the test routine should be written to carry out the following sequence of functions.

1 Read data from cards, set up a TAB and associated areas, and the Service TAB HDRSCST.

2 Optionally, record the initial state of these areas on the line printer.

3 Pass control to the user's Peripheral Monitor routine, by branching to HDRPMCUE. Peripheral Monitor will hence be able to service the request set up in the TAB, and access suitable test files where necessary. It will

return control to the test routine by branching to HDRSCCUE.

4 On each entry from Peripheral Monitor, record the State of the TAB, associated areas, Service TAB and any other relevant areas.

5 Return to step 1 to initiate a further test.

Test routines such as that described could incorporate such features as off-lining of test output to disc for input to Selective Print, and other facilities similar to those offered by General Bead Tester.

A further testing routine might be written to test the completed Driver program without using communications terminals. Such a routine would replace the communications handling routines, and would access a card reader, line printer or files on backing storage in place of the communications terminals. In response to an MPPUT macro, for example, this communications simulator routine might output data to a line printer. Simularly, trial data could be read in from a card reader in response to an MPGET macro. Other macros could be largely dealt with by updating the housekeeping reply words with suitable parameters - also read in from the card reader.

A routine of this type could additionally be used to independently test a user written communications monitor routine before link-testing the complete Driver program.

Chapter 8

Driver Testing Aids

#XJBA
#XJBB

NAME

#XJBA (magnetic tape)
#XJBB (direct access)

VERSION

Mark 1

TITLE

Trial Data Set Up (TDSU)

HARDWARE REQUIREMENT

6720 words of main store (#XJBA)
7488 words of main store (#XJBB)
1 card reader or 1 paper tape reader
1 line printer (120 chs. minimum)
1 or more more magnetic tape decks (#XJBA)
1 or more direct access storage units (#XJBB)

EXECUTIVE PRIORITY

50

DESCRIPTION

General

TDSU sets up or amends a file held on magnetic tape (#XJBA) or on a direct access device (#XJBB). Parameters submitted by the user on cards or paper tape control the run and specify the data to be written to the file. The file created may have any required file identifier and records may be set up

in any format provided that their length does not exceed 1000 words.

When used as part of the Testing Aids suite, the purpose of TDSU is to create trial data records for input to General Bead Tester. In this case the length of each record set up must not exceed 510 words. There is no reason, however, why the program should not be used independently to create or amend other types of file, in which case the 1000 word limit on output record length applies.

Two modes of operation are available to the user: *set up mode* and *amendment mode.*

SET UP MODE

A complete direct access or magnetic tape trial data file is set up according to the parameters submitted to the program. These parameters consist of *control parameters*, defining the run mode required and the characteristics of the file to be set up, followed by *input data parameters* containing the data to be written to the file.

AMENDMENT MODE

An existing trial data file of like characteristics to the file to be produced (i.e.same device type, bucket/block sizes, record sizes etc.) can be amended. The unit of amendment is a record. The old file is copied to the new file, stopping at each point where records are to be inserted, deleted or replaced.

The data required for the amendment is submitted on cards or paper tape and consists of control parameters (as described above, but also defining the characteristics of

the old file) together with *input amendment parameters* specifying the alterations to be made to the file. Input data parameters are also submitted at each point where new data is to be generated for the file.

Input

Input to the program for a set up run consists of control parameters and input data parameters.

Input for an amendment run consists of a magnetic tape or direct access storage unit holding the old trial data file to be amended, together with control parameters as above, any input data parameters required and input amendment parameters.

Output

Main output consists of a trial data file for submission to General Bead Tester. The contents of this file are described in the section *The Trial Data file* on page 32. The program will also output a line printer listing of the parameters submitted to it, any error in an individual parameter being indicated by an error code accompanying the parameter image. This listing is described in the section *Line printer output* on page 42.

CONTROL PARAMETERS

Each control parameter is punched as a separate card or paper tape record, starting at column 1. The parameters are described below in the order in which they must be submitted.

Run description parameter

The run description parameter specifies the type of run required (set up or amendment) and the characteristics of the trial data file to be created.

FORMAT

Set up run

The format of the run description parameter for a set up run is

TSDU,V*xxxx*,*yyyy*
or
TDSU,F*xxxx*,*yyyy*

where

TDSU indicates that a set up run is required

V indicates that variable length trial data records are to be created

F indicates that fixed length trial data records are to be created

xxxx is the maximum bucket/block size of the trial data file to be created.

For a magnetic tape trial data file the size specified must be ≤ 1024 words

For a direct access trial data file, the size specified may be 128, 256, 512 or 1024 words. Values less than four characters in length must be augmented by zeroes on the left hand side e.g. 0256, 0512.

yyyy is the maximum output record size in words.

For a magnetic tape trial data file this must be ⩽ 510 words. For a direct access trial data file the size must be

⩽ (*xxxx* - 2)
or
510 words

whichever is the smaller.

Note: Where a direct access trial data file is to be created, it is important that the maximum bucket size *xxxx* agrees with the bucket size of the output file to be opened. The file's bucket size will have been defined previously using the file allocator program #XPJC. A description of this program can be found in the 1900 *Unified Direct Access Standards Utilities* Manual, Chapter 2.

Amendment run

The format of the run description parameter for an amendment run is

TDAM,V*xxxx*,*yyyy*
or
TDAM,F*xxxx*,*yyyy*

where TDAM indicates that an amendment run is required, and all other parameter fields are as described above.

IN

This parameter is only required for an amendment run, in which case its use is mandatory. The parameter provides the

information required by the program in order to open a trial data file which is to be amended.

BASIC FORMAT

In its basic form, an IN parameter will simply specify the name of the file to be amended, thus:

IN(*oldfile*)

where *oldfile* is the name of the old trial data file to be amended.

EXTENDED FORMATS

If required, for example where several generations of a trial data file have been previously created, the user can specify the file to be amended in more detail. The full format is as follows:

For a file held on magnetic tape:

INV(*oldfile(fgn/rsn),tsn*)

where

fgn is the generation number of the file.

rsn is the reel sequence number

tsn is the tape serial number, in octal. The number may be punched with or without a preceding * or #.

For a direct access trial data file:

INV(*oldfile(fgn),csn*)

where

fgn is the file generation number of the file.

csn is the cartridge serial number, in octal. The number may be punched with or without a preceding * or #.

The user may either include all these additional parameter fields, as shown above, or he may punch any one or any combination of them, for example:

IN(*oldfile(/rsn),tsn*)
IN(*oldfile,tsn*)
IN(*oldfile(fgn)*)

OUT

This parameter provides the information required by the program to open the output file.

BASIC FORMAT

In its basic form, an OUT parameter will simply specify the name of the file to be opened, thus:

OUT(*newfile*)

where *newfile* is the name of the output file to be opened. It will hence also be the name of the trial data file created in the course of the run unless a RENname parameter is used (page 10).

EXTENDED FORMAT

If the file to be opened is held on magnetic tape, the user may additionally specify its file generation number *fgn*, reel sequence number *rsn* and tape serial number *tsn* as described f

the IN parameter (page 7). The full format of an OUT parameter for a magnetic tape file will thus be:

OUT(*newfile(fgn/rsn)*,*tsn*)

As in the case of the IN parameter, the user may punch any one or any combination of these additional fields.

The full format of an OUT parameter to open a direct access file is as follows:

OUT(*newfile(fgn1 = fgn2)*,*csn*)

where

fgn1 is the existing file generation number of the file.

fgn2 is the new generation number to be given to the file when it is opened. Permitted range is 0 to 4095 inclusive.

Note: If *fgn1* is omitted, the highest generation of the file will be opened and its generation number altered to that specified in *fgn2*.

If *fgn2* is omitted, the file generation number specified as *fgn1* will remain unchanged.

If both *fgn1* and *fgn2* are omitted, the file will be given file generation number 4095.

csn is the cartridge serial number, in octal. The number may be punched with or without a preceding * or # and must be in the range #0 to #777777

As in the case of the IN parameter, the user may either include all these fields, or any one or any combination of them, for example:

OUT(*newfile,csn*)
OUT(*newfile(=fgn2),csn*)
OUT(*newfile(fgn1)*)

RENname

This parameter may optionally be included in order to rename the file *newfile* specified in the OUT parameter.

BASIC FORMAT

In its basic form, a RENname parameter will simply specify the new name to be given to the output file when it is opened, thus:

REN(*newname*)

where *newname* is the new name to be given to the file (up to 12 alphanumeric characters, hypen or spaces, starting with an alphabetic character).

EXTENDED FORMAT

As with the previous control parameters, the user may give further details if these are required. The full format of the RENname parameter is as follows:

For a file held on magnetic tape:

REN(*newname(fgn/rsn),retn*)

where

fgn is the file generation number to be given to the renamed file. Permitted range is 0 to 4095: if this

field is omitted, 0 is assumed.

rsn is the reel sequence number to be given to the renamed file. Permitted range is 0 to 4095: if this field is omitted, 0 is assumed.

retn is the retention period in days to be allocated to the tape on which the file is held. Permitted range is 0 to 4095: if this field is omitted, 4095 is assumed.

For a direct access file:

REN(*newname*,*fgn*/*vn*)

where

fgn is the file generation number to be given to the renamed file. Permitted range is 0 to 4095: if omitted, 0 is assumed.

vn is the version number to be given to the renamed file. Permitted range is 0 to 4095: if omitted, 0 is assumed.

As implied above, the user may specify any one or any combination of these additional fields, provided that the fields specified are relevant to the type of file (magnetic tape or direct access) in use. For example:

REN(*newname*,*retn*)
REN(*newname*(/*vn*))
REN(*newname*(*fgn*),*retn*)

PEND parameter

This parameter signifies the end of the control parameters.

FORMAT

PEND

INPUT DATA PARAMETERS

Introduction

The input data parameters submitted to TDSU specify trial data to be written to the output file. In a set up run, all the data to be written to the output file is included in these parameters. In an amendment run, they may be used in conjunction with input amendment parameters (page 29) in order to insert new records into an existing trial data file.

The TDSU output file produced from these parameters will be used as input to General Bead Tester (GBT). To avoid possible confusion this file will normally be referred to as the trial data file. Where necessary, it will be called the 'new' trial data file in order to distinguish it from the 'old' trial data file input to TDSU during an amendment run.

The trial data file created in a TDSU run will contain the data required by GBT to set up one or more TABs (one per bead or logical sequence of beads to be tested) and simulate Driver services. The input data parameters submitted to TDSU must therefore specify the contents of each TAB to be set up by GBT, together with the data to be inserted into the associated areas of these TABs in order to simulate Driver services.

A number of data definitions and directives, collectively known as *items*, are available for the user to create a trial data file according to his precise requirements. The remainder of this section describes how these items are used to generate and output different types of data to the trial data file. The purpose, format and sequence of the individual trial data records to be assembled from this data is described in the section *The trial data file* on page 32.

General Description

An input data parameter consists of a single card or paper tape record and its maximum length is therefore 80 characters. The data required to generate a trial data record is punched as a series of contiguous fields within one or more parameters. Each of these fields is one item, and is distinguished from the preceding and succeeding items in the parameter by the type of data it contains. The user preparing input data parameters must start a new item each time that the type of data to be written to the trial data file changes.

An item may contain any one of the following types of data

Record header
Alphanumeric character(s)
Decimal value(s)
Octal value(s)
Sterling value(s)
Start new bucket/block indicator
End-of-input indicator

The first item required to create a trial data record is a *record header* item. When read in, this item indicates to the program that the previous trial data record (if any)

assembled in main store is now to be written to the trial data file, and that creation of a new trial data record is to begin. Thereafter, the subsequent items in the parameters are dealt with in the order that they appear, generating the required data in consecutive locations of main store until the complete output record has been assembled. On detection of the next record header item the record is written to the trial data file and the process is then repeated for each trial data record required until a complete trial data file has been set up.

Each of the input data parameters required to create a particular trial data record may comprise one or more items, the only restriction being that an item must not run on from one parameter to the next. The maximum length of an item is thus 80 characters. If space permits, the last item in each parameter should be terminated by a space character ∇. The user is free to use any character positions to the right of the terminating space characters for his own comment: this will be output on the line printer listing together with the parameter image and will provide a useful reference when examining the listing.

The full format of an input data parameter may thus be given as

Column 1 *Column 80*
↓ ↓
item-1 item-2 ..... item-n∇user comment

Items

OVERALL FORMAT

An item is made up of several subfields, which are punched contiguously in the following order.

1 Repetition factor (optional)

2 Data type

3 Implied length of data expression (optional)

4 Beginning of data expression indicator

5 Data expression

6 End of data expression indicator

Repetition factor

Use of the repetition factor relieves the user of the tedious and error-prone task of writing out and punching repetitive sets of like type data in full. If present, it will cause the subsequent data expression to be repeatedly inserted into consecutive locations within the output record currently being created. The maximum number of repetitions that can be specified is 9: if the data is only to be generated once the repetition factor is omitted.

Data type

This subfield is used to hold a directive indicating the type of data to be generated. Since items are classified according to the type of data they contain, the directive also indicates item type.

Directive	*Item/data type*
R	Record header
H	Characters
#	Octal data

Directive	*Item/data type*
D	Decimal data
£	Sterling data
NEW▽	Start new bucket/block
END▽	End of input data

Each item type is further described below (page 17 onwards).

Implied length

Use of this subfield is optional. If present, it consists of a three digit value (permitted range 002 to 999) which specifies the length in words of the output field to be generated from the subsequent data expression.

If the data expression is smaller than the implied length of the output field, the program will generate sufficient space characters (in the case of character items) or zeroes (all other items) to increase the length of the data expression to the implied length. This process is known as *padding:* the spaces or zeroes will appear in the output field to the right of the data expression.

Padding is particularly useful in cases where user data to be inserted is much shorter than the length of the field to be set up. For example, the user specifying the contents of name and address fields need not include the non-significant spaces or zeroes needed to bring each field up to its required length: these will be automatically inserted by the program.

If the data expression is longer than its implied length (for example, due to a data preparation error) the excess characters at the right hand side of the data expression will be ignored.

Note that any padding or truncation of individual data expressions will occur without any warning message being output on the line printer.

Beginning and end of data expression indicators

A data expression must always be enclosed in apostrophes, thus:

'*data expression*'

Data expression

A data expression comprises data to be written to the trial data file. Its format is dependent on the type of item in which it appears.

Within a character item the data expression consists of an uninterrupted string of characters to be written to the trial data file. The program will insert these characters, four at a time, into consecutive words of the trial data record currently being created. Within other items, the data to be written to the trial data file is punched one word at a time, each word being seperated by a comma.

Item types

RECORD HEADER ITEM

This indicates to the program that all succeeding items up to the next record header item are to be used to generate a complete trial data record.

There are two types of record header item, one for fixed length records and one for variable length records. The difference between them is that for fixed length files the

user does not specify the trial data record length, the program using the information provided in the Run Description parameter (page 5).

The format of a record header item is as follows:

If a fixed length trial data record is to be formed:

R

For a variable length record:

R'*xxxx*'

where *xxxx* is a data expression specifying the length of the record to be created in words. The value given must be punched to the right of the data expression, any unused character positions being left as leading spaces or filled with zeroes. For example, a 256 word trial data record would be specified as follows:

Either:

R'V256'

or:

R'0256'

The record header item will always be the first item in the first parameter to create a trial data record, and the character R must always appear in column 1 of this parameter.

Character items

A character item is declared as a string of up to 77 characters (the data expression) enclosed by two apostrophes and preceded by the directive #. Any of the 64 characters of

the internal machine code set may be specified in the data expression, the only proviso being that an apostrophe character (') to be written to the trial data file must be represented by two adjacent apostrophes (") within the string.

The characters within the data expression will be inserted, four characters at a time, into consecutive words of the trial data record. Any unfilled character positions in the last word of trial data created will be spacefilled automatically.

An alphanumeric character item may be punched in any of the following formats

*r*H*nnn*'*string*'
*r*H'*string*'
H*nnn*'*string*'
H'*string*'

where

r is the repetition factor (permitted range 2 to 9)

H indicates data type (characters)

nnn is the implied length (permitted range 002 to 999)

string is a data expression comprising a string of characters to be inserted into the output record. Up to 77 characters may be specified, the limitation being the size of the parameter within which the item is held (maximum length 80 characters).

Padding with space characters or truncation to satisfy implied length requirements takes place at the right hand side of the data expression before any repetition occurs.

Examples

Starting at a word *n* of the trial data record currently being formed

Item *Trial data generated*

Word	*n*	*n*+1	*n*+2	*n*+3	*n*+4	*n*+5
H'EXAMPLE∇'	EXAM	PLE∇				
H'EXAMPLE'	EXAM	PLE∇				
H004'EXAMPLE'	EXAM	PLE∇	∇∇∇∇	∇∇∇∇		
4H'BUZZ'	BUZZ	BUZZ	BUZZ	BUZZ		
3H'OFF'	OFF∇	OFF∇	OFF∇			
2H'JONES'	JONE	S∇∇∇	JONE	S∇∇∇		
2H003'JONES'	JONE	S∇∇∇	∇∇∇∇	JONE	S∇∇∇	∇∇∇∇
H'THIS∇IS∇AN∇APOST ROPHE∇''∇'	THIS	∇IS∇	AN∇A	POST	ROPH	E∇'∇
H'THIS∇IS∇AN∇APOST ROPHE∇'''	THIS	∇IS∇	AN∇A	POST	ROPH	E∇'∇
H004'THIS∇IA∇AN∇A POSTROPHE∇'''	THIS	∇IS∇	AN∇A	POST		
H002''''''∇''''''''	'''∇	''''				
2H003''''''∇'''''' ''''	'''∇	''''	'∇∇∇	'''∇	''''	'∇∇∇
H006'PAD'	PAD∇	∇∇∇∇	∇∇∇∇	∇∇∇∇	∇∇∇∇	∇∇∇∇
H002'TRUNCATE∇THIS'	TRUN	CATE				
6H'∇'	∇∇∇∇	∇∇∇∇	∇∇∇∇	∇∇∇∇	∇∇∇∇	∇∇∇∇

DECIMAL ITEM

The data expression within a decimal item consists of a signed or unsigned decimal value for each word of decimal data to be

generated by the item. Each decimal value is seperated from the next by a comma; the complete set of values forming the data expression is enclosed in apostrophes and preceded by the directive D.

The decimal values are stored as binary numbers in consecutive words of the output record. Unsigned values are taken to be positive and any significant space characters are taken to be zeroes.

A decimal item may be punched in any of the following formats:

$rDnnn'd_1,d_2, \ldots\ldots d_n'$

$Dnnn'd_1,d_2, \ldots\ldots d_n'$

$rD'd_1,d_2, \ldots\ldots d_n'$

$D'd_1, d_2, \ldots\ldots d_n'$

where

r	is the repetition factor (permitted range 2 to 9)
D	indicates the data type (decimal)
nnn	is the implied length of the data expression (permitted range 002 to 999)
$d_1,d_2,\ldots d_n$	is the data expression and consists of the decimal values, seperated by commas, to be inserted into the next n words of the trial data record. Each decimal value specified must be an integer within the range $\pm 2^{23}$ Up to 39 decimal values may be specified, the limitation being the size of the parameter within which the item is held (maximum length

80 characters). If only one decimal value is specified, it must not be followed by a comma.

Padding with zeroes or truncation to satisfy implied length requirements takes place at the right hand side of the expression before any repetition occurs.

Examples

Starting at a word *n* of the trial data record currently being formed:

Item	*Trial data generated*							
Word	*n*	*n*+1	*n*+2	*n*+3	*n*+4	*n*+5	*n*+6	*n*+7
D'1'	+1							
D'4,5,6'	+4	+5	+6					
D'+1'	+1							
D'-1'	-1							
D'+4,5,+6'	+4	+5	+6					
D004'1,-2,3'	+1	-2	+3	0				
4D'6,-3'	+6	-3	+6	-3	+6	-3	+6	-3
2D004'+1,+2,+3'	+1	+2	+3	0	+1	+2	+3	0
8D'0'	0	0	0	0	0	0	0	0
8D'-1'	-1	-1	-1	-1	-1	-1	-1	-1
4D002'5,6,7,8'	+5	+6	+5	+6	+5	+6	+5	+6
2D002'-5761,43'	-5761	+43	-5761	+43				
3D'-2,+4'	-2	+4	-2	+4	-2	+4		
D008'16'	+16	0	0	0	0	0	0	0
D008'16,15,-2'	+16	+15	-2	0	0	0	0	0

Item	*Trial data generated*							
Word	*n*	*n*+1	*n*+2	*n*+3	*n*+4	*n*+5	*n*+6	*n*+7
D'∇,16,∇∇14'	0	+16	+14					
D008'∇'	0	0	0	0	0	0	0	0
8D'∇'	0	0	0	0	0	0	0	0

OCTAL ITEM

The data expression within an octal item contains a string of up to eight octal digits for each word of octal data to be generated. Each string of octal digits is seperated from the next by a comma: the complete set of octal strings forming the data expression is enclosed by two apostrophes and is preceded by the directive #.

Each consecutive pair of octal digits occupies one character position in the trial data record being created. The data is stored right justified within a word. If a string contains an odd number of digits, the program automatically inserts a zero before the first digit. If generation of octal data stops in the middle of a word of the trial data record, the remainder of the word is automatically filled with zeros. Leading space characters within a string, or complete strings of space characters, are all taken to be zeroes.

An octal item may be punched in any of the following formats

r#nnn'string-1,string-2,.....string-n'

r#'string-1,string-2,.....string-n'

#nnn'string-1,string-2,.....string-n'

#'string-1,string-2,...string-n'

where

r	is the repetition factor (permitted range 2 to 9)
#	indicates the data type (octal)
nnn	is the implied length of the data expression (permitted range 002 to 999)
string-1, *string-2*,... ...*string-n*	is the data expression, and consists of the strings of octal digits, seperated by commas, to be inserted into the next *n* words of the trial data record. Up to 39 words of octal data may be specified, the limitation being the size of the parameter within which the item is held (maximum length 80 characters). If only one octal string is specified, it must not be followed by a comma.

Examples

Starting at a word *n* of the trial data currently being formed:

Item *Trial data generated*

Word *n*	*n*+1	*n*+2	*n*+3
#'1234567' 01 23 45 67			
2#'01234567' 01 23 45 67	01 23 45 67		
#'0123' 00 00 01 23			
#'012 00 00 00 12			
#002 '6' 00 00 00 06	00 00 00 00		

Item	*Trial data generated*			
Word	*n*	*n*+1	*n*+2	*n*+3
2#002 '345'	00 00 03 45	00 00 00 00	00 00 03 45	00 00 00 00
#002 '01234567, 01234567, 01234560'	01 23 45 67	01 23 45 67		
2#002 '1212 3434, 45607, 123'	12 12 34 34	00 04 56 07	12 12 34 34	00 04 56 07
3#'012'	00 00 00 12	00 00 00 12	00 00 00 12	
2#'14621467, 0'	14 62 14 67	00 00 00 00	14 62 14 67	00 00 00 00
4#'0'	0	0	0	0
#'▽'	0			
# '▽▽▽▽1234'	00 00 12 34			
4#'▽'	0	0	0	0
#004 '01234567, ▽; ▽▽2	01 23 45 67	0	2	0
#' '	0			
4#' '	0	0	0	0
# ',,777'	0	0	00 00 07 77	0

STERLING ITEM

The data expression within a sterling item contains a signed or unsigned value in pounds and pence for each word of sterling data to be generated. Each value is seperated from the next by a comma: the complete set of sterling values forming the data expression is enclosed by two apostrophes and preceded by the directive £.

Each sterling value is converted to new pence and is then stored in the trial data record in exactly the same way as

a decimal value. Unsigned values are taken to be positive. Significant space characters are taken as zeroes unless the value is signed: in the latter case at least one digit, 0 if necessary, must be used to specify the pounds part of the value and must precede the full stop (see below).

A sterling item may be punched in any of the following formats

r£*nnn'pounds.pence-1,pounds.pence-2,.....pounds.pence-n'*

r£*'pounds.pence-1,pounds.pence-2,.....pounds.pence n'*

£*nnn'pounds.pence-1,pounds.pence-2,.....pounds.pence-n'*

£*'pounds.pence-1,pounds.pence-2,...pounds.pence-n'*

where

r	is the repetition factor (permitted range 2 to 9)
£	indicates data type (sterling)
nnn	is the implied length (permitted range 002 to 999).
pounds.pence-1, pounds.pence-2, ...pounds. pence-n	is the data expression and consists of sterling values, seperated by commas, to be inserted into the next *n* words of the output record. For each value, *pounds* must be specified as one or more decimal digits. *Pence* must always consist of two decimal digits; ½p cannot be specified. Up to 39 sterling values may be specified, the limitation being the size of the parameter within which the item is held

(maximum length 80 characters). If only one sterling value is specified, it must not be followed by a comma.

Padding with zeroes or truncation to satisfy implied length requirements takes place at the right hand side of the data expression before any repetition occurs.

Note: A sterling value will be stored in a trial data record as a binary integer representing the decimal number of new pence: the full stop separating pounds and pence is omitted in the trial data record. If convenient, the user may therefore specify any sterling value as a decimal value within a decimal data expression.

For example:

£'356.06'

and

D'35606'

will both cause the decimal value 35606 to be inserted into a word of the trial data record as a binary value.

Examples

Starting at a word *n* of the trial data record currently being formed:

Item *Trial data generated*

Word	*n*	*n*+1	*n*+2	*n*+3
£'10.16'	+1016			
£'-10.16'	-1016			
£'∇∇∇10.16	+1016			

Item	*Trial data generated*			
Word	*n*	*n*+1	*n* +2	*n* +3
£'+16.10,4.19, -2.36,100.00'	+1610	+419	-236	+10000
£003'100.00'	+10000	0	0	
2£002'100.00'	+10000	0	+10000	0

NEW ITEMS

The user may wish to arrange his trial data file so that, for ease of access, certain records are each stored in the first position of a bucket or block.

A NEW item is punched in a seperate input data parameter and causes the next output record specified in the subsequent parameters to be stored in the first position of the next available bucket/block.

A NEW item must be punched in columns 1 to 4 of the input data parameter in which it appears, and always has the format

NEW∇

The parameter must not contain any other item.

The next parameter(s) will normally consist of an R item followed by whatever other items are required to generate the first record for the new bucket/block. However, if required, further NEW items, one per parameter, may be punched after the initial NEW item, causing the equivalent number of buckets/blocks to be skipped in the trial data file. A NEW item may also precede an END item (see below) or any of the input amendment parameters (page 29).

END ITEM

This item indicates that there is no more input data for the file currently being set up or amended. If applicable, the last trial data record created is written to the output file. If, in an amendment run, the end of the old trial data file has not been reacted, its remaining records are copied to the output file. Statistical information is then output to the line printer. All files are closed and the program is deleted.

The END item must be punched in columns 1 to 4 of the input data parameter in which it appears, and always has the format

END▽

Input amendment parameters

Input amendment parameters enable the user to edit records input from the old trial data file to be amended. Each parameter is submitted as a seperate card or paper tape record.

The unit of amendment is one trial data record. Each record within the old trial data file is uniquely identified by an 8-digit number of the format

aaaabbbb

where

aaaa is the bucket/block number of the record, starting at 0001 for the first bucket/block on the file.

bbbb is the record sequence number, starting at 0001 for the first record in each bucket/block.

Any trial data records not specified in the input amendment parameters are copied directly to the output file.

DELETE

The DELETE parameter can be used to delete one record, a sequence of records, or from a specified record to the end of the old trial data file. The formats required in each case are as follows:

Format	*Result*
DELETE▽*aaaabbbb*	Record *aaaabbbb* is not copied to the new trial data file.
DELETE▽$aaaa_1bbbb_1$,$aaaa_2bbbb_2$	Records $aaaa_1bbbb_1$ to $aaaa_2bbbb_2$ inclusive are not copied to the new trial data file.
DELETE▽*aaaabbbb*,END▽	Record *aaaabbbb* and all subsequent records in the old trial data file are not copied to the new trial data file.

INSERT

An INSERT parameter is always followed by one or more input data parameters. Copying of the old trial data file is temporarily halted at record specified in the INSERT parameter; one or more new records are then generated from the input data parameters and are copied to the new trial data file in the normal way. Copying of the old trial data file then proceeds, starting with the record specified in the INSERT parameter.

An INSERT parameter thus specifies a record in the old trial data file before which one or more new records are to be inserted. It may also be used to insert records at the end of the file. The alternative formats are as follows.

Format	*Result*
INSERTV*aaaabbbb*	The new output records specified in the subsequent input data parameters are inserted before record *aaaabbbb*.
INSERTVENDV	Any remaining records on the old trial data file are copied to the output file. The new trial data records specified in the input data parameters following INSERTVENDV are added to the end of the file.

REPLACEMENTS

If a DELETE parameter (page 30) is followed by input data parameters, the trial data records generated from the latter will be inserted in the new trial data file starting at the location that would have been occupied by the first deleted record. A record may thus be replaced by submitting a DELETE parameter followed by the input data parameter(s) to generate a new trial data record.

PARAMETER TERMINATOR RECORDS

The user may set up or amend several trial data files in a single run by submitting several sets of input parameters. The END parameter for each set must be separated from the

control parameters of the next set by two parameter terminator records. These records will cause the program to suspend at the end of a run and output a console message indicating that it is ready to be restarted.

The two terminator records are punched as follows

	Column 1 ↓
First record:	****
Second record:	Blank

THE TRIAL DATA FILE

The trial data file created by TDSU contains all the trial data required for one GBT run. In order to set up one file, the user must submit a set of control parameters (page 4) followed by the input data parameters to create the following trial data records, in sequence.

1 A *File Header record*

2 For each bead test path (i.e. one test on a bead or a logical sequence of beads):

(a) A *TAB Header record*, followed if necessary by one or more *TAB Continuation records*

(b) One or more *Request Header records*, in each case followed by one or more *Request Continuation records* if necessary.

3 An *end-of-file* record.

The purpose and format of these records is described below.

File header record

DESCRIPTION

This record indicates beginning-of-file.

LAYOUT

Word 0	Record length in words (always 5)
Word 1	Record identification (HVVV)
Words 2 to 4	Filler data (all zeroes)

TAB Header record

DESCRIPTION

A TAB Header record, together with TAB Continuation records, if required, defines the contents of a TAB and its associated areas, with the exception of the TAB's request area. The TAB areas set up by GBT from this data will be used for one bead test path: since all manipulation of these areas will be carried out by GBT and the beads in the test path it is only necessary to specify the contents of the areas once. Thus only one TAB Header record is required for each test path. As the maximum size of a trial data record is 510 words, any data in excess of this length which is required to define a TAB must be set up in one or more TAB Continuation records following the TAB Header record.

LAYOUT

Word 0	Record length (≤510)
Word 1	Record identification (TVVV)

Word 2 — Message identification. This consists of a 4-character identifier allocated by the user to the bead test path. The identifier may be used for selecting records for printing during a run of Selective Print and may consist of any four characters except END▽.

Word 3 — Contents of logical terminal number field of the TAB.

Word 4 — Bead number of first bead to be entered in the bead test path.

Words 5 to 9 — The values to which accumulators 4 to 7 and 0 respectively are to be set on entry to the first bead in the bead test path.

Word 10 — Contents of the permanent priority field of the TAB.

Word 11 onwards — 1 Associated store cell definitions (for GBT dynamic store simulation only).

There are parameters for associating one Input/Output and/or one Additional Core area with the TAB for initial entry to the bead test path.

A two-word entry is required in each case, as follows:

Word 0 Area Type:

IO▽A indicates store cell used as Input/Output area

ACA▽ indicates store cell used as Additional Core area

Word 1 The block number of the imaginary block from which simulated allocation of this cell will take place under GBT. This number is known as the *Chain link number*: for further details see the section *Dynamic store simulation* in the GBT specification.

2 Area data definitions

The remainder of the record consists of these definitions, which specify the data to be held in the Input/Output, Message, and Additional Core areas associated with the TAB on entry to the bead test path.

Each area data definition is set up as follows

Word 0 Area type:

MAVD indicates Message area

IOAD indicates Input/Output area

ACAD indicates Additional Core area

Word 1 The displacement in words from the beginning of the specified area to the first word of user data. This must be >4 as the first four words of the area will

be reserved to conform with Driver standards.

Word 2 — The length of the data submitted for insertion in the area, in words.

Word 3 onwards — Data to be held in the area.

TAB Continuation records

DESCRIPTION

If the area data definition for a TAB and associated areas cannot all be contained in one TAB Header record, excess data follows in one or more TAB Continuation records. Note that neither TAB Header nor TAB Continuation records need be filled to capacity and thus the contents of different areas can be defined as separate records in the trial data file. This method of file organization is extremely convenient if it later becomes necessary to amend the file.

LAYOUT

Word 0 — Record length (≥5, ≤510)

Word 1 — Record identification (TC▽▽)

Word 2 onwards — Data continuing from where it was left off in the previous record. Data may be continued from one record to another at any point.

Request Header records

DESCRIPTION

The Request Header records specify the various requests expected from the beads in the bead test path, in the order that these requests are expected to be issued. Each record specifies one expected request, and indicates to which bead, if any, control is to be passed following that request. If a peripheral input request is expected, it also holds further data which GBT will insert into the Input/Output area of the TAB to simulate servicing of the request.

The maximum length of a Request Header record is 510 words. If this is not sufficient to hold all the data required (that is, a relatively large peripheral input transfer is to be simulated) the data is continued on one or more Request Continuation records.

LAYOUT

Word 0 Record length (≥5, ≤510).

Word 1 Record identification (R∇∇∇).

Word 2 Request code expected to be issued by a bead at the point in the bead test path for which the record is submitted.

Word 3 Identity of the next bead to be entered, or four space characters ∇∇∇∇ if the bead test path is to be terminated.

Word 4 If peripheral input is not to be simulated, four zeroes 0000.

If peripheral input is to be simulated, the length of the data held in Word 5 onwards.

Word 5 onwards — Data to be placed in the Input/Output area of the TAB to service the expected peripheral input request.

Request Continuation records

DESCRIPTION

If the data required to service a peripheral input request cannot all be held on the appropriate Request Header record, it is continued on one or more Request Continuation records. Neither Request Header or Request Continuation records need be filled to capacity before continuing data on the next record.

LAYOUT

Word 0 — Record length, in words (≥5, ≤510)

Word 1 — Record identifier (RCVV)

Word 2 onwards — Further data

End-of-file record

DESCRIPTION

The End-of-file record concludes the trial data file.

LAYOUT

Word 0	Record length (always 5 words)
Word 1	Record identifier (ENDV)
Words 2 to 4	Filler data (all zeroes)

OPERATING INSTRUCTIONS

In the following narrative, #XJB*n* signifies either #XJBA or #XJBB, depending on which version of the program is being used.

	Narrative	*Console Message*
1	Load #XJB*n*	
2	Load the required magnetic tape decks (#XJBA) or direct access storage units (#XJBB)	
3	To activate the program:	
	(a) for parameters on paper tape, input:	GO#XJB*n* 20
	(b) for parameters on cards, input:	GO#XJB*n* 21
4	(a) If the run has been successful, the program will output the message	HALTED:- OK
	If a further run is required, continue from stage 2 above.	
	(b) If the run has been unsuccessful due to an error in the control parameters, the program will	

Narrative	*Console Message*
output the message	HALTED:- PROGRAM TERMINATED
If a further run is required, input:	GO#XJB*n* 23
(c) If it becomes necessary to abandon the current run for one of the reasons listed under Exceptions conditions page 41, either of the following actions may be taken:	
1 If no further runs are required abandon the program,	
2 If the program is to be restarted for another run, input:	GO#XJB*n* 23
(d) If the run has been unsuccessful for any other reasons, a post mortem of the run may be obtained by inputting	GO#XJB*n* 24
On completion of the post mortem, the program will output the message	HALTED:- PM
The program may then be abandoned or restarted as in (c) above.	
5 If the program has been restarted by GO#XJB*n* 23, it will indicate that it is ready to commence the next run by outputting message	HALTED:- NE
Continue from stage 2 above.	

Exception conditions

Message	*Reason*	*Action*
DISPLAY:- NEW FILE EXTENDED	New trial data file has been extended by 80 blocks	None required
HALTED:- BS	Bucket size of new trial data file does not equal that described in the Run Descriptor control parameter (XJBB only)	GO #XJBB or abandon current run
HALTED:- CE	Card reader error	GO #XJB*n* or abandon current run
HALTED:- ED	File not successfully renamed (XJBB only)	Abandon current run
HALTED:- HK*nn*	Housekeeping exception condition: *nn* gives the contents of word 11, character 0.	GO #XJB*n* or abandon current run
HALTED:- LE	Line printer error	GO #XJB*n* or abandon current run
HALTED:- LOAD EDS**nnnnnn*	File opened in wrong cartridge (#XJBB only)	Put correct cartridge #*nnnnnn* lowest on line and GO#XJBB

Message	*Reason*	*Action*
HALTED:- LOAD TSN**nnnnnnnn*	File opened on wrong tape (#XJBA only)	Put correct tape #*nnnnnnnn* lowest on line and GO#XJBA
HALTED:- MT	Failure to open specified file (#XJBA only)	Abandon current run
HALTED:- TE	Tape reader error	GO #XJB*n* or abandon current run

LINE PRINTER OUTPUT

The printed output from Trial Data Set Up is basically a listing of all the input data and input amendment parameters submitted to the program together with any program generated reports concerning the input.

The line

TRIAL DATA SET UP *dd/mm/yy*

is output at the beginning of the run, *dd/mm/yy* being the date of the run.

If the control parameters are valid, the different types of information given in these parameters (run type, file names etc.) are output against the appropriate headings as one or more lines of print. If a control parameter contains an error, two lines are output to the printer:

1 The image of the parameter in error

2 The appropriate error code (see page 46)

Two lines are then output to act as heading lines for the listing of the input parameters, thus:

```
RECORD          INPUT RECORD IMAGE          REPORT
NUMBER CH.1                                 ITEM CODE
```

The significance of these headings for each parameter listed is as follows.

RECORD NUMBER — is the sequence number of the parameter as it occurs within the raw data input (one parameter per record).

CH.1/INPUT RECORD IMAGE — Each parameter/record image is printed out under the INPUT RECORD IMAGE heading, starting immediately below the CH.1 heading

Nothing will be printed out under the remaining headings unless the parameter listed contains an error. In this case, the headings have the following significance

REPORT

- ITEM — The number printed out under this heading is in effect the sequence number of the item in error within the parameter, reading from left to right. For example, if a parameter image is followed by the figure 3 in the ITEM column, this indicates that the third item from the left is the item in error.
- CODE — An error code is printed out under this heading, and indicates the type of error which has occurred. A full list of error codes is given on page 46.

Note: Vetting of a parameter stops as soon as an error is encountered. Each parameter containing an error should therefore be carefully checked for subsequent errors before

being re-submitted.

Warning lines: output record padded or truncated

TSDU always sets up trial data records of the size specified by the user in the Run Descriptor parameter (files with fixed length records) or Record Header items (files with variable length records). If the total length of the data submitted to form a trial data record (including data expressions individually padded by the program) is greater or less than the specified length of the record, the record will be padded or truncated accordingly before being written to the trial data file.

If a trial data record has been padded the listing of the parameters to set up that record will be immediately followed by the line

OUTPUT RECORD PADDED ***

If the record has been truncated, the line

OUTPUT RECORD TRUNCATED ****

is output.

CLOSEDOWN FORCED error message

If, during an amendment run, input amendment parameters are submitted in the wrong order, the program will ignore all parameters submitted for the current amendment after the first misplaced parameter.

Closedown is initiated when the program encounters an input amendment parameter specifying a trial data record which has already been deleted or copied to the new trial data file. The parameter causing the error is listed, followed by the

CLOSEDOWN FORCED BECAUSE OF AN ERROR IN ABOVE RECORD

and

FOLLOWING PARAMETERS IGNORED

The remaining parameters for the file are then listed, followed by the end reports for the run (see below).

End reports

Four statistical reports are output at the end of each run, as follows:

1 Number of card or paper tape records read in, including the END record.

2 Number of valid records output to the new trial data file. In an amendment run, this includes records copied across from the old file.

3 Number of buckets or blocks output to the new trial data file (includes end-of-file bucket).

4 Number of card or paper tape records containing errors.

Post Mortems

The operator can obtain a post mortem of an unsuccessful run at the point of failure, using entry point 24 (see Operating Instructions, page 40). The program will output the following information on the line printer.

1 State of accumulators

2 All lower data areas

3 The program area

4 Contents of housekeeping buffers

5 Contents of output buffer for new trial data file.

Error codes

ERRORS IN CONTROL PARAMETERS

Invalid Run Description parameter

Error code	*Reason*
A	First card or paper tape record is not a Run Description parameter
B	Characters 5 or 11 not comma (,)
C	Character 6 not F or V
D	Characters 7 to 10 not numeric
E	Specified block size exceeds 1024 words (#XJBA)
	Specified bucket length not 128, 256, 512 or 1024 words (#XJBB)
F	Characters 12 to 15 not numeric
G	Invalid block size (#XJBA)
	Invalid record length (#XJBB)
H	Character 16 not space (∇)

Other control parameter errors

Error code	*Reason*
J	IN parameter not found when expected
K	OUT parameter not found when expected
L	RENname or PEND parameter not found when expected
M	Invalid filename
N	Invalid file generation number or Invalid tape serial number (#XJBA) or Invalid cartridge serial number (#XJBB)
O	Tape serial number not octal (#XJBA) or Cartridge serial number not octal (#XJBB)
P	Tape serial number exceeds #37777777 (#XJBA) Cartridge serial number exceeds #777777 (#XJBB)
Q	Separator character not comma(,)

Invalid input data/input amendment parameters

Error code	*Reason*
01	Invalid item type
02	Repetition factor outside permitted range (2 to 9)

Error code	*Reason*
03	Implied length outside permitted range (002 to 999)
04	Beginning of data expression indicator (') not found
05	End of data expression indicator (') not found
06	Non-numeric characters found in numeric data expression
07	Invalid sign (i.e. not + or -)
08	Separator within data expression not comma (,)
09	Input parameters do not terminate with an END item
10	First item in an input data parameter is invalid because the preceding parameter is INSERT or DELETE or consists of a NEW item. A Record Header item is required.
20	Record length specified in a Record Header item exceeds maximum record size specified in the Run Descriptor control parameter
40	Octal string is more than 8 characters in length
41	Octal digit outside range 0 to 7
43	Pounds field not 1 to 5 characters in length
44	Pence field less than 2 characters in length

Error code	*Reason*
50	Decimal value not within range $\pm 2^{23}$
63	Sterling value outside range - £83886.08 to + £83886.07
80	Bucket/block and record number specified in an input amendment parameter is invalid
81	Second bucket/block and record number specified in a DELETE parameter is invalid
83	Bucket/block and record number specified in an input amendment parameter is out of sequence
84	Second bucket/block and record number specified in a DELETE parameter is out of sequence
85	Insertion of data suppressed because of error in INSERT parameter

SDAD

SDAT

SUBROUTINE GROUP NAME

SDAD (direct access)

SDAT (magnetic tape)

VERSION

Mark 1

TITLE

General Bead Tester (GBT)

COMPONENT ROUTINES

HDRGBTINTM (SDAT) HDRGBTINTD (SDAD)
HDRGBTTSU
HDRGBTBS
HDRGBTRA
HDRGBTCDOWN
HDRGBTPMTM
GCOREPRINT

HARDWARE REQUIREMENT

5K words of main store (not including store occupied by beads under test, GBT Master routine, and user defined constants and tables).

1 or more direct access storage units (SDAD)

2 magnetic tape decks (SDAT)

1 card reader or paper tape reader

DESCRIPTION

General

General Bead Tester is a set of routines which enables beads to be tested individually or in groups within a batch processing environment. Beads can be tested without the need to amend any of their constituent instructions or statements, and without any of the files or communication devices required for real time operation being on-line.

Bead testing is carried out by running the beads to be tested under GBT instead of Driver. A bead testing program incorporating GBT is assembled by consolidating the GBT routines with the beads to be tested, the GBT Master routine, and magnetic tape or direct access housekeeping, as appropriate. The GBT Master routine is a simplified version of the single-threading Driver Master routine and is described in the section *General Bead Tester Standards*, page 55.

The unit of testing under GBT is a *bead test path* consisting of a bead or logical sequence of beads to be tested. During the operation of the test program GBT initiates and monitors

the execution of one test path at a time. In each case, parameters and data for processing by the beads in the path are initially set up in a TAB and the appropriate associated area(s) respectively, using trial data input from a file previously set up in a TDSU run. GBT then causes the first bead in the path to be entered: thereafter it passes control between the beads in the path in exactly the same manner as Driver and simulates all other Driver services requested by them. The reaction of the beads to the trial data is recorded by writing the contents of the TAB and associated areas to an output file on each entry to and exit from a bead This file may subsequently be edited and printed out for analysis by the programmer using the Selective Print program.

Method of operation

As explained in the description of TDSU (page 32) the trial data submitted to GBT for each bead test path to be executed in a particular run consists of a TAB Header record and Request Header records, each record being followed by continuation records where necessary.

GBT uses each TAB Header record to perform the Driver simulation required for initial entry to a bead test path. Input of a TAB Header record first causes GBT to write a *Bead Test Path Entry* record to the output file: this record gives the bead test path's message identification as defined in the TAB Header record, and hence provides a convenient point of reference when the file is subsequently printed out. GBT then uses the remaining trial data contained in the TAB Header record to set the TAB and associated areas for entry to the first bead in the test path. The TAB is thus set to request entry to the appropriate bead, whilst the associated areas are set to contain data of the type that will be presented to the bead during its operation in the live on-line program. Before passing control to the specified bead, GBT writes the

current contents of the TAB and associated areas to the output file in the form of first *TAB input record* for the path. A further TAB input record is thereafter generated and output each time a bead is entered or re-entered during the subsequent execution of the test path.

Each time a bead issues a request, a Request Header record is read in from the trial data file. The expected request from the bead, as specified in the Request Header record, is compared with the request actually issued by the bead. Provided that the actual and expected requests are the same, the contents of the TAB and associated areas on exit from the bead are written to the output file in the form of a *TAB output record*.

The main information provided by GBT during successful execution of a bead test path hence consists of alternate TAB input and TAB output records. If an error occurs, for example a mismatch between actual and expected requests, execution of the bead test path is terminated and a *Bead Test Path Exit* record is output giving the reason for termination. This is followed by a *'Record ignored' record* for each record read in from the trial data file until a new TAB Header record is encountered. If no error occurs, a Bead Test Path Exit record indicating successful termination of the test path is output.

Before and/or during the execution of each test path, GBT simulates Driver servicing in the following four ways.

1 Communications input is simulated by setting up the Message area of the TAB before entry to the test path according to data contained in the TAB Header record submitted for the path.

2 Peripheral input is simulated by either:

(a) Setting up the Input/Output area in the same manner as the Message Area before entry to the bead test path.

(b) Using trial data held in the Request Header and any Request Continuation records submitted for each print in the run where peripheral input requests are expected. The trial data held in the record(s) submitted for each peripheral input request is read into the Input/Output area of the TAB.

3 GBT reacts to requests for peripheral or communications output by outputting the current contents of the TAB and associated areas (i.e. generating a TAB output record in the normal manner) and passing control to the next bead. Output requests are thus in effect ignored. However, since entry to any bead other than the Error Recovery bead implies that the preceding request has been successfully serviced, the execution of the beads in the test path is uneffected.

A request for TAB deallocation always causes termination of the current bead test path, even if, due to a programming error, the request also specifies transfer of control to another bead.

4 Dynamic store allocation and deallocation may be simulated under GBT, although GBT only supports single threading operation and hence uses a fixed store system. Where store requests are issued in a bead test path, the Input/Output and Additional Core areas allocated in the Master routine are used by the beads instead of the requested cells.

Although GBT does not allocate or deallocate store, it does carry out a number of validity checks on store requests. Full details are given in the section

Dynamic store simulation, on page 57 .

Input

Main input to GBT consists of a magnetic tape (SDAT) or direct access (SDAD) trial data file previously set up in a TDSU run. Parameters submitted on cards or paper tape control the GBT run.

Output

Output consists of a GBT output file on magnetic tape (SDAT) or disc (SDAD) ready for input to Selective Print.

GENERAL BEAD TESTER STANDARDS

Store organization

The following standards apply:

1 As in all versions of Driver, a Message area of the maximum size required must be permanently allocated to the TAB.

2 Since GBT works in single threading mode, the Input/Output and Additional Core areas, if required, must also be permanently allocated to the TAB and must be of the maximum size required during the run.

3 Where dynamic store allocation is to be simulated, Driver standards must be followed in that:

(a) Store requests for allocation of two cells of store must request these cells in ascending order

of block number

(b) A maximum of one Input/Output area and one Additional Core area may be requested during the processing of a message.

GBT Master routine

A GBT Master Routine is essentially a simplified version of the Master routine required by the single threading Driver. An additional table is required if dynamic store allocation is to be simulated. The program name and priority of the complete program will be as defined by the user in this routine.

DEFINING THE TAB AND ASSOCIATED AREAS

Within a GBT Master routine, the TAB and associated areas are defined under the following common blocknames.

Area	*Location*
TAB	HDRGBTTAB
Message area	HDRGBTMA
Input/Output area	HDRGBTIOA
Additional Core area	HDRGBTACA

Words 22 and 23 of the TAB must be preset as follows:

	Fixed store operation only	*Simulated dynamic store allocation*
Word 22	Start address of Input/Output area	-1
Word 23	Start address of Additional Core area	-1

Word 0 of the Message, Input/Output and Additional Core areas must in each case be set to contain the length in words of the area.

USER - DEFINED CONSTANTS AND TABLES

All programs

The following constants and tables must be defined as described for the single-threading Driver Master routine in Chapter 4.

1 Highest bead number constant (HDRBDCONST)

2 Bead branch table (HDRBT1)

3 Facility code constant (HDRRACONST)

Dynamic store simulation

In order to carry out simulated store allocation and deallocation and vet beads requests for these services, the only information required by GBT is the size of the cells that will be set up in each block of the common pool for the operational Driver program.

This requirement is met by setting up a table of one word entries, one entry per block, under the common area blockname HDRSAQ. The entry for each block gives the length in words of each cell that will be defined within that block when the common pool is set up. The entries must be defined in ascending order of block number: for example, to simulate the presence of the common pool described in Chapter 2 (page 22), the locations HDRSAQ to HDRSAQ+3 would be set as follows

Location	*Contents*	
HDRSAQ	256	(Block 0)
HDRSAQ+1	500	(Block 1)
HDRSAQ+2	64	(Block 2)
HDRSAQ+3	100	(Block 3)

As mentioned in the section *Store requests*, Chapter 3, a bead requesting allocation of a store cell by Driver does not explicitly specify the length of cells required, but instead gives the number of the block comprising cells of the required size. This block number, otherwise known as the chain link number, is used by GBT as a modifier in order to access the appropriate entry in the table and hence determine the size of cell requested.

Request codes

The following request codes cause action to be taken by GBT in addition to its normal functions.

TAB DEALLOCATION

If TAB deallocation (bit 10 = 1) is specified, the bead test path is terminated.

PERIPHERAL INPUT

If the request code is found to have the facility code 1, and the Request Header and any Request Continuation records submitted for the request contain trial data for processing

the data is placed in the Input/Output area of the TAB starting at the location specified by the displacement value in word 3 of the TAB.

STORE REQUESTS

Request codes 3010 and 3020 (allocate and deallocate store respectively) cause GBT to initially check whether the program has been set up to perform dynamic store simulation. GBT will then check to ensure that requests for store allocation and deallocation conform to the following standards.

Allocation requests

1 Allocation of a store cell or cells must not have been previously requested during the execution of the bead test path.

2 The size of cell requested for use as a particular area type (i.e. as an Input/Output or an Additional Core area) must not exceed the maximum length defined for that area in the GBT Master routine.

3 The destination addresses specified in the request must be 23 or 24.

Double allocation requests

1 Where two cells are requested for allocation, the cells must be requested in ascending order of block number. In other words the block number for the first cell requested (word 1 of the TAB) must be less than the block number of the second cell requested (word 3 of the TAB).

2 The destination address in word 2 of the TAB must be different from that in word 4 of the TAB.

The destination address(es) of the cell(s) to be deallocated must be 23 or 24.

Deallocation requests

As (2) above.

Overlay

No overlay facilities exist within General Bead Tester and hence all the beads under test must be store resident at run time.

Entry to General Bead Tester

The magnetic tape version of GBT (subroutine group SDAT) is entered from the GBT Master routine on one of the following instructions.

If the input parameters are on paper tape:

```
BRN HDRGBTINTIM
```

If the input parameters are on cards:

```
BRN HDRGBTINTIM+1
```

Similarly, the direct access version (subroutine group SDAD) is entered as follows:

Input parameters on paper tape:

```
BRN HDRGBTINTD
```

Input parameters on cards:

BRN HDRGBTINTD+1

CONTROL PARAMETERS

Up to four control parameters are required for each GBT program run. Each parameter is punched as a seperate card or paper tape record, starting at column 1. The parameters are described below in the order in which they must be submitted.

IN

This parameter provides the information required by the program in order to open the input (trial data) file.

BASIC FORMAT

In its basic form, an IN parameter will simply specify the name of the input file thus:

IN(*filename*)

where *filename* is the name of the input file

EXTENDED FORMATS

If required, for example where several generations of trial data file have been previously created, the user can specify the file in more detail. The full format is as follows:

For an input file held on magnetic tape:

IN(*filename*(*fgn/rsn*),*tsn*).

where

fgn is the generation number of the file.

rsn is the real sequence number

tsn is the tape serial number, in octal. The number may be punched with or without a preceding * or #.

For a direct access input file:

IN(*filename*(*fgn*),*csn*)

where

fgn is the file generation number of the file.

csn is the cartridge serial number, in octal. The number may be punched with or without a preceding * or #.

The user may either include all these additional parameter fields, as shown above, or he may punch any one or any combination of them, for example:

IN(*filename*(/*tsn*),*tsn*)

IN(*filename*,*tsn*)

IN(*filename*,*fgn*))

OUT

This parameter provides the information required by the program to open the output file.

BASIC FORMAT

In its basic form, an OUT parameter will simply specify the name of the output file to be opened, thus:

OUT(*newfile*)

where *newfile* is the name of the file to be opened. It will hence also be the name of the GBT output file created in the course of the run unless a RENname parameter is used (page 64).

EXTENDED FORMAT

If the file to be opened is held on magnetic tape, the user may additionally specify its file generation number *fgn*, reel sequence number *rsn* and tape serial number *tsn* as described for the IN parameter (page 61/62). The full format of an OUT parameter for a magnetic tape file will thus be:

OUT(*newfile*(*fgn/rsn*),*tsn*)

As in the case of the IN parameter, the user may punch any one or any combination of these additional fields.

The format of an OUT parameter to open a direct access file is as follows:

OUT(*newfile*(*fgn1=fgn2*),*csn*)

where

fgn1 is the existing file generation number of the file.

fgn2 is the new generation number to be given to the file when it is opened. Permitted range is 0 to 4095 inclusive.

Note:- If *fgn1* is omitted, the highest generation of the file will be opened and its generation number altered to that specified in *fgn2*

If *fgn2* is omitted, the file generation number specified as *fgn1* will remain unchanged

If both *fgn1* and *fgn2* are omitted, the file will be given file generation number 4095.

csn is the cartridge serial number, in octal. The number may be punched with or without a preceding * or # and must be in the range #0 to #777777.

As in the case of the IN parameter, the user may either include all these fields, or any one or any combination of them, for example:

OUT(*newfile*,*csn*)

OUT(*newfile*(=*fgn2*),*csn*)

OUT(*newfile*(*fgn1*))

RENname

This parameter may optionally be included in order to rename the file *newfile* specified in the OUT parameter.

BASIC FORMAT

In its basic form, a RENname parameter will simply specify the new name to be given to the output file when it is opened, thus:

REN(*newname*)

where *newname* is the new name to be given to the file (up to 12 alphanumeric characters, hyphen or spaces, starting with an alphabetic character).

EXTENDED FORMAT

As with the previous control parameters, the user may give further details if these are required. The full format of the RENname parameter is as follows:

For a file held on magnetic tape:

REN(*newname*(*fgn/rsn*),*retn*)

where

fgn is the file generation number to be given to the renamed file. Permitted range is 0 to 4095: if this field is omitted, 0 is assumed.

rsn is the reel sequence number to be given to the renamed file. Permitted range is 0 to 4095: if this field is omitted, 0 is assumed.

retn is the retention period in days to be allocated to the tape on which the file is held. Permitted range is 0 to 4095: if this field is omitted, 4095 is assumed.

For a direct access file:

REN(*newname*,(*fgn/vn*))

where

fgn is the file generation number to be given to the renamed file. Permitted range is 0 to 4095: if omitted, 0 is assumed.

vn is the version number to be given to the renamed file. Permitted range is 0 to 4095: if omitted, 0 is assumed.

As implied above, the user may specify any one or any combination of these additional fields, provided that the fields specified are relevant to the type of file (magnetic tape or direct access) in use - for example:

REN(*newname*,*retn*)

REN(*newname*(/*vn*))

REN(*newname*(*fgn*),*retn*)

PEND

This parameter signifies the end of the control parameters.

FORMAT

PEND

OPERATING INSTRUCTIONS

	Narrative	*Console Message*
1	Load the program	
2	Load the required magnetic tape decks or direct access storage units	—
3	To activate the program, input: where *program name* is the name of the test program and *nn* is the chosen entry point.	GO *program name nn*
	The program will output:	DISPLAY:- GENERAL BEAD TESTER MK *nnnn type*
	where *nnnn* is the mark number of the GBT subroutine group and *type* is either TAPE or DISC depending on whether group SDAT or SDAD is present within the program.	
4	(a) If the run has been successful, the program will output the message:	HALTED:-OK
	If a further run is required, continue from stage 2 above.	
	(b) If the run has been unsuccessful due to an error detected by GBT, the program will output:	DISPLAY:-*error message* HALTED:-ZZ

	Narrative	*Console Message*
	The action to be taken is described in the section *Exception conditions* below.	
5	If a post-mortem of the run is required, input:	GO*program-name*25
	The program will output the message: and abandon the run.	HALTED:-PM
6	To abandon the run without taking a post-mortem, input:	GO*program-name*24
	The program will halt with the message:	HALTED:-OK

Exception conditions

Message	*Reason*	*Action*
DISPLAY:-GENERAL BEAD TESTER-WRONG INPUT FILE HALTED:-ZZ	Word 1 of first record of input file is not HVVV,i.e. first record of the file is not a valid file header record	Abandon the run
DISPLAY:-OUTPUT FILE EXTENDED	Output file has been extended by 80 blocks (SDAD only)	None required

Message	*Reason*	*Action*
HALTED:-LOAD EDS*nnnnnnnn	File opened on wrong cartridge (SDAD only)	Put correct cartridge *#nnnnnnnn* lowest on line and GO *program name*
HALTED:-LOAD TSN*nnnnnnnn	File opened on wrong tape (SDAT only)	Put correct tape #nnnnnnnn lowest on line and GO *program name*

#XJBC

#XJBD

NAME

#XJBC (magnetic tape)

#XJBD (direct access)

VERSION

Mark 1

TITLE

Selective Print

HARDWARE REQUIREMENT

5248 words of main store (#XJBC)

5056 words of main store (#XJBD)

1 card reader or 1 paper tape reader

1 line printer (120 chs. minimum)

1 or 2 magnetic tape decks (#XJBC)

1 or 2 direct access storage units (#XJBD)

EXECUTIVE PRIORITY

#XJBC: 70

#XJBD: 50

DESCRIPTION

General

The function of the program is to edit and print the output produced by General Bead Tester. Parameters submitted by the user on cards or paper tape specify which records and fields within records are to be printed from a GBT output file on magnetic tape (#XJBC) or disc (#XJBD).

Two modes of operation are available: select mode and compare mode.

SELECT MODE

Select mode enables complete records and individual fields within records to be selected for printing from a GBT output file. Parameters specifying the records whose contents are to be printed out are submitted in the order that these records appear within the file.

COMPARE MODE

If one or more beads in a bead test path are amended following the initial run under GBT, it will often be necessary to re-submit the same input data to the test path to determine whether amendments to an individual bead or beads within the path have been successful. A second and possibly further GBT

runs using the same input data may thus be required. Use of the compare mode of Selective Print reduces the time required to check the results of each run against those of the previous one.

In compare mode, the latest file produced by GBT in respect of a particular series of bead test paths is compared with that produced in a previous run. Differences in file header records are ignored, since these always differ between one run and another. However, data following the file header record is compared on a record for record basis. As soon as a discrepancy is detected, compare mode ceases and the program goes into select mode. The remainder of the latest GBT output file is printed out selectively or in full, as directed by the input parameters.

Input

Input to the program for a select run consists of the GBT output file to be printed together with parameters held on cards or paper tape. The parameters consist of a set of *Header records* which describe the run, followed by sets of Select and Edit records and concluding with an end record.

For a compare run, input is as above with the addition of a further GBT output file produced during a previous test. This file is known as the *compare file*.

Output

Output consists of a line printer listing of the data selected from the GBT output file, together with reports on any errors detected during the GBT run and during input to Selective Print.

HEADER RECORDS

Input data submitted to the program on cards or paper tape begins with two or three header records, depending on the type of run. The records each contain a control parameter, and are described below in the order that they are submitted. All records are punched starting at Column 1.

Run description

Either:

SELECT (for a select run)

or: COMPARE (for a compare run)

INA

For #XJBC:

INA(*newfile*(*fgn*),*tsn*)

For #XJBD:

INA(*newfile*(*fgn*),*csn*)

where

newfile is the name of the GBT output file.

fgn is the generation number of the file. It may be omitted, in which case the brackets enclosing it must also be omitted.

tsn is the tape serial number of the file. It may be omitted, in which case the preceding comma must be omitted.

csn is the cartridge serial number of the file. It may be omitted, in which case the preceding comma must also be omitted.

INB (Compare run only)

For #XJBC:

INB(*oldfile*(*fgn*),*tsn*)

For #XJBD:

INB(*oldfile*(*fgn*),*csn*)

where *oldfile* is the name of the compare file and all other parameters are as described for INA.

SELECT AND EDIT RECORDS

The header records submitted to Selective Print are immediately followed by the first Select record and its associated Edit records. Each Select record submitted to the program can select GBT output records for printing in one of three ways.

1 Select all records

2 Select records by message identification

3 Select records by record type

In each case, editing of the specified records is carried out according to the Edit records following the Select record.

Select records

The format and use of the records required to carry out the three types of selection mentioned above are described below.

SELECT ALL RECORDS

Format

SELECTVALL

Use

The GBT output file is printed from its current position; each GBT output record being edited where required according to the Edit records following the Select record. Note that no further sets of Select and Edit records can be submitted in this case, because Selective Print reads the file serially.

SELECT RECORDS BY MESSAGE IDENTIFICATION

Record format

Either: SELECTVMSG = *aaaa*

or SELECTVMSG = *aaaa*,*bbbb*

or SELECTVMSG = *aaaa*,END

where *aaaa* and *bbbb* are message identifications.

Use

The User will have assigned a 4-character message identification to each bead test path executed by GBT. The identification for each test path is set up in the course of generating the TAB Header record for the path during a TDSU run (see TDSU specification, page 34).

On reading in a Select record in one of the above formats, Selective Print skips through the GBT output file until the Bead Test Path Entry record containing the specified message identification is found. According to the format of the submitted select record, Selective Print will then either

1 Select all GBT output records produced by execution of bead test path *aaaa*.

2 Select all GBT output records produced by execution of all bead test paths from *aaaa* up to and including the Bead Test Path Entry record for bead test path *bbbb*.

3 Select all records in the file starting at the Bead Test Path Entry record for bead test path *aaaa*.

In each case, the GBT output file is printed the specified position according to the Edit records following the Select record.

SELECT RECORDS BY RECORD TYPE

Format

Either:- SELECT▽REC=*xxxx*

or:- SELECT▽REC=$xxxx_1$,.....$xxxx_n$

where *xxxx* is the record identification which was generated by GBT when producing the output record(s) selected for printing. Up to 4 types of GBT output record may be selected using the second format shown above. The identifiers for the various types of record are as follows.

Record type	*Record identification*
Bead Test Path Entry recor	SVVV
TAB input records	TlVV
TAB output records	T0VV
Bead Test Path Exit records	EVVV
'Record ignored' records	IVVV

Use

Selective Print skips through the GBT output file until a record of the specified type is found. This record, and all other records of the required type(s) are printed out according to the Edit records currently controlling printout until the file is exhausted. Note that no further sets of Select and Edit records can be submitted in this case, because Selective Print reads serially through the file.

Edit records

Edit records specify which fields of the selected GBT output records are to be printed. Since each set of input parameters (one Select record plus any required Edit records) are dealt with as a unit, the order in which Edit records occur within each set is of no importance.

For editing purposes, the fields within GBT output records are classified into the following six types:

1	Request area of TAB (words 0 to 8)	Fields within TAB input and TAB output records
2	Remainder of TAB (words 9 to 27)	
3	Message area	
4	Input/Output area	
5	Additional Core area	
6	All other fields	All fields within Bead Test Path Entry, Bead Test Path Exit, and Record Ignored records)

An Edit record which is submitted to control the printing of a particular GBT output record or set of records can specify any one of the following three *print options*.

1 *Print data in full*. In this case, all records specified by the preceding Select record will be printed in full.

2 *Contracted printing*. In this case, the printing action taken depends on the type of field being edited, as follows.

Type of field	*Action*
Request area of TAB	Printed in full
Remainder of TAB	

Type of field	*Action*
Message area Input/Output area Additional Core area	Only words 0 to 23 of the specified field are printed out. Any further data is ignored.
All other fields	Not printed.

The main benefit of this option is that it considerably reduces printing time by avoiding the need to print out the full contents of the message, Input/Output and Additional Core area fields in each TAB input and TAB output record selected.

3 *No printing.* Where some fields within the selected records are not required for analysis, the Edit records submitted for them may specify that these fields are not to be printed.

The format and use of the records required to edit the various types of GBT output record are described below.

EDIT ALL SELECTED RECORDS

Format

Either:- EDITVALL

for full printing of all records specified in the preceding Select record,

or:- EDITVALL(CON)

for contracted printing of all selected record.

Use

The record specifies the action to be taken on all GBT output records selected by the preceding Select record.

EDIT TAB INPUT RECORDS

Format

EDITvT1vFIELD = *field-1*(*qualifier-1*),*field-2*(*qualifier-2*),...
....*field-n*(*qualifier-n*)

where:-

field-1, *field-2*,... *field-n* are each 2-character *area mnemonics* specifying which fields of the selected TAB input records are to be printed. The mnemonics for each type of field are as follows.

RA	Request area
TF	Remainder of TAB
MA	Message area
IO	Input/Output area
AC	Additional Core area

qualifier-1, *qualifier-2* *qualifier-n* are statements specifying the print option required in each case. These statements may be either:

CON	(for contracted printing of the field).
or	
NO	(for no printing of the field).

Note that:

1 If no area mnemonic is supplied for a particular type of field, the field will not be printed.

2 If only one type of field is specified the comma is omitted from the end of the *field∇qualifier* expression.

Use

An Edit record of this type must be submitted when selection of TAB input records is specified by Select record. This must be done irrespective of whether selection is implicit (i.e. by message identification) or explicit (by record type).

EDIT TAB OUTPUT RECORDS

Format

EDIT∇TO∇FIELD = *field-1(qualifier-1),field-2(qualifier-2),*
....field-2(qualifier-n)∇

where the supplied parameters are as described in the previous section.

Use

As for TAB input records (see previous section).

EDIT REST OF DATA

Format

Either:-

EDITVREST

for full printing of the selected records

or:-

EDITVREST(NO)

where printing of the records is not required

Use

An Edit record of this type must be submitted when selection of any type of record other than TAB input and TAB output records is specified by a Select record either implicitly or explicitly. All Bead Test Path entry, Bead Test Path exit, and 'Record Ignored' records selected by the preceding Select record will be printed in full or ignored, as required.

END RECORD

Format

END

Use

This record is the last input parameter submitted for the run. It must be followed by the standard end records i.e. one record with four asterisks (****) in columns 1 to 4 followed by either a blank card (input on cards) or two newline characters (input on paper tape).

OPERATING INSTRUCTIONS

In the following narrative, #XJBx signifies either #XJBC or #XJBD, depending on which version of the program is in use.

	Narrative	*Console Message*
1	Load #XJBx	
2	Load the required magnetic tape deck(s) (#XJBC) or direct access storage unit(s) (#XJBD)	
3	To activate the program:	
	(a) For parameters on paper tape, input:	GO#XJBx 20
	(b) For parameters on cards, input:	GO#XJBx 21

Narrative	*Console Message*
4(a) If the run has been successful, the program will output the message:	HALTED:-HH
If a further run is required, input:	GO#XJB*x* 25
The program will indicate that it is ready to restart by outputting the message:	HALTED:-HH
(b) If the run has been unsuccessful, the program will output:	DISPLAY:- *error message* HALTED:- ZZ
Either abandon the run immediately or first initiate a post-mortem by inputting:	GO#XJB*x* 27
The completion of the post-mortem is indicated by the message:	HALTED:- SP
The program cannot be restarted in this case.	

Exception conditions

Message	*Reason*	*Action*
DISPLAY:- PARAMETER ERROR *n*	Errpr in input header record,	Abandon run.

Message	*Reason*	*Action*
	as defined by the error code *n* (see *Error codes* page 86).	
DISPLAY:-SELECTIVE PRINT - WRONG DATA HALTED:-ZZ	First 8 characters of first input Header record are not SELECT▽▽ or COMPARE▽	Abandon run
DISPLAY:-SELECTIVE PRINT - WRONG COMPARE FILE HALTED:-ZZ	Word 1 of first record in compare file is not H▽▽▽	Abandon run
DISPLAY:-SELECTIVE PRINT - WRONG SELECT FILE	Word 1 of first record in GBT output file is not H▽▽▽	Abandon run
HALTED:- CE	Card reader error	GO#XBJ*x* or abandon run
HALTED:- HK#*nn*	Housekeeping exception condition: #*nn* gives the error code. For details, see the section *Exception Conditions* of the manual *Direct Access*, page 129, or Table 11, page 108 of the Manual *Magnetic Tape*, as appropriate.	GO#XJB*x* or abandon run

Message	*Reason*	*Action*
HALTED:- LOAD EDS *nnnnnn	File opened on wrong direct access storage unit (#XJBB only)	Put correct storage unit #*nnnnnn* lowest on line and GP#XJBB)
HALTED;-LOAD TSN *nnnnnnnn	File opened on wrong tape (#XJBC only)	Put correct tape #*nnnnnnnn* lowest on line and GO#XJBC.
HALTED:-LP	Line printer not available	Make line printe. available and GO#XJB*x*
HALTED:-PL	Paper low condition on line printer	Load more paper and GO#XJB*x*
HALTED:-TE	Tape reader error	GO#XJB*x* or abandon run
HALTED:-TF	Line printer error	GO#XJB*x* or abandon run

ERROR CODES

On detection of an error in an input header record, the program displays the message PARAMETER ERROR *n* where *n* is an error code and halts ZZ.

The error codes are as follows:

Error Code	*Meaning*
A	No INA record when expected

Error Code	*Meaning*
B	No INB record when expected
C	Invalid filename
D	Generation number *fgn* > 4095 (#XJBC) or Reel Serial Number *rsn* > 511 W#XJBD)
E	Non octal character in cartridge serial number *csn* (#XJBC) or Non octal character in tape serial number *tsn* (#XJBD)
F	*csn* > #7777777 (#XJBC) or *tsn* > #37777777 (#XJBD)
G	Invalid or missing seperator character

LINE PRINTER OUTPUT

Line printer output consists of run heading lines, followed by an edited listing of the selected GBT output records, together

with error messages where necessary.

Run heading lines

Printed output from Selective Print begins with a line giving the program name (XJBC or XJBD) its version number and the date of the run. This is followed by the line:

SELECTIVE PRINT OF THE OUTPUT FROM THE RUN OF GENERAL BEAD TESTER DATED *dd/mm/yy*

where *dd/mm/yy* is the date of the GBT run.

In a select mode run, printing of the selected GBT output data now commences.

For a run in compare mode, the line

COMPARE MODE

is printed at this point. No printing of GBT output data occurs unless a discrepancy is detected between the GBT output and Compare files, whereupon the program prints the line

DISCREPANCY FOUND:-RECORD TYPE *xxxx* MESSAGE IDENTITY *aaaa*

where

xxxx is the record identification (4 characters or spaces)

aaaa is the relevant message identification (4 characters or spaces)

Printing then continues in select mode.

Printing of GBT output records

BEAD TEST PATH ENTRY RECORDS

if this type of record is selected for printing, the line

BEAD TEST PATH - MESSAGE IDENTIFICATION *xxxx*

is printed each time such a record is found, *xxxx* being the message identification for the bead test path in each case.

TAB INPUT AND TAB OUTPUT RECORDS

If any fields in either of these types of record are to be printed, the appropriate subheading line is first output on selection of each record, i.e.

TAB INPUT

or

TAB OUTPUT

Request area fields

If the request area fields of TAB Input and/or TAB Output records are to be printed, each field will be printed out in the following format.

Line 1:- REQUEST AREA WORD 0 1 2 3 4 5 6 7 8

Line 2:- The contents of each word as their character representation, in each case printed below the appropriate number.

Remainder of TAB fields

If required, each of these fields is printed out over four lines, thus:

```
Line 1  TAB:        TERM NO   ENTRY PT.  THIS BEAD  NEXT BEAD  SPARE         ACCUM-
                                                                              ULATORS
                    xxxx      +nnnnnnnn  +nnnnnnnn  +nnnnnnnn  xxxxxxxxxxxx  00000000
        5           6         7
        00000000    00000000  00000000

Line 2  QUEUE LINK  MESSAGE AREA  I/O AREA  WORK AREA  PERM.PRIORITY  TEMP.PRIORITY
        00000000    00000000      00000000  00000000   +00000000      +00000000
```

Where *xxxx* is the contents of the appropriate word printed as their character representation

+nnnnnnnn is the contents of the appropriate word printed as a signed decimal integer

oooooooo is the contents of the appropriate word printed in octal

Message area fields

If Selective Print prints out either the first four words of each message area field required (contracted printing) or the entire contents of each field (full printing). Each field is printed in the following format.

Line 1:- MESSAGE AREA: WORDS 0-3: *nnnnn/xxxxx/xxxxx/xxxxx* MSG. LENGTH = *+nnnnnnnn*

where *+nnnnnnnn* and *xxxx* are as described for Rest of TAB fields

Line 2:- Message area data (words 4 to 23)

Line 3:- Contents of the remainder of the field, printed at 20 words per line.
onwards
(full
print
only)

Input/Output area fields

Fields of this type are each printed out in the following format

Line 1:-IO AREA: WORDS 0-3: *nnnnn/xxxx/xxxx/xxxx*

where *nnnnn* and *xxxx* are as described for Rest of TAB fields

Line 2:-Contents of words 4 to 23 of Input/Output area field.

Line 3:-Contents of the remainder of the field, printed at *onwards* 20 words per line
(full
print
only)

Additional core area fields

Fields of this type are printed out in the same format as Input/Output area fields, with the difference that Line 1 begins with the title ADDITIONAL CORE AREA.

BEAD TEST PATH EXIT RECORDS

Where this type of record is selected, the line

BEAD TEST PATH - EXIT BECAUSE *reason*

will be printed following selection of each such record. The reason given will be

SUCCESSFUL END OF TEST PATH

if the record was output by GBT following execution of a complete test path. Otherwise the reason given will take the form of one of the error messages described in the section *Line printer error messages* on page 95.

'RECORD IGNORED'RECORDS

When an error in trial data records input to GBT has caused 'Record Ignored' records to be written to the GBT output file, the latter type of record can be selected for printing as described on page 77. 'Record Ignored' records are printed out one per line in the format

RECORD IGNORED:- MESSAGE*aaaa*RECORD*xxxx*REQUEST*nnnn*

aaaa is the message identification (word 2 of the record)

xxxx is the record identification (word 3 of the record)

nnnn is the request code of the ignored record (word 4)

Each of the above are printed as four characters.

End of run

At the end of the run, the line

END OF SELECTIVE PRINT RUN

is output.

Line printer error messages

Message	*Reason and/or action*
1 BEAD TEST PATH-EXIT BECAUSE	
(a) BEAD NOT IN SYSTEM-END TEST	CALL instruction for next bead to be entered not present in bead table HDRBT1. Amend the GBT Master routine accordingly.
(b) BRANCH TO BEAD IS INVALID	Request for entry to another bead also specifies deallocation of TAB and is therefore invalid. Amend the offending bead accordingly.
(c) CONTROL INFORM-ATION MUTILITATED	Words 0 to 3 of the Message, Input/Output or Additional Core area have changed between entry to and exit from a particular bead, and have therefore been corrupted by the bead.
(d) INVALID INPUT DATA	An error has been found in a TAB Header or Request Header record input to GBT during execution of a bead test path. The error may have occurred for one of several reasons, depending on the type of record.
	Tab Header records
	(i) Store cell definition submitted for fixed store run.

Message	*Reason and/or action*
	(ii) More than one store cell definition has been submitted for a particular type of area (i.e. more than one cell defined for use as an Input/Output area or as an Additional Core area).
	(iii) An inconsistency in store cell or area data definitions has resulted in GBT failing to identify an item in a TAB Header record.
	(iv) Data submitted for insertion into a particular store cell exceeds maximum length of the cell as specified in the Input/Output or Additional Core area definitions in the GBT Master routine.
	(v) An area data definition has set up user data in words 0 to 3 of the area.
	(vi) In an area data definition, the length of the data set up exceeds the specified length of the area.
	Note that items (ii) and(iv) above apply only to GBT runs performing simulated dynamic store allocation.

Message	*Reason and/or action*
	Request records
	(i) Length of submitted data exceeds specified record length.
	(ii) Length of submitted data less than specified record length.
	The required format for both the above types of record is described in the TDSU specification, pages 33 to 36.
INVALID REQUEST	This condition arises when either:
	(i) A store allocation or de-allocation request is issued by a bead when GBT is being run using a fixed store system.
	(ii) Servicing of a peripheral input request by GBT causes any or all of words 0 to 3 of the Input/ Output area to be overwritten (i.e. invalid displacement).
	(iii) An attempt is made to allocate the same type of area to a TAB more than once (for example, two attempts are made to allocate a store cell to a TAB for use as an Input/Output area).
	(iv) An error has occurred in a store allocation request due to:

Message

Reason and/or action

(a) A bead requesting access to store cell blocks in the wrong order (see Chapter 2, page 23).

(b) Incorrect destination address (i.e. not binary 23 or 24 - see GBT specification page 59).

(c) Specified length of a cell requested from an imaginary block is greater than the length specified for Input/Output or Additional Core areas in GBT Master routine, depending on the purpose for which the cell is requested.

(v) An error has occurred in a single deallocation request because either

(a) The source address is not binary 23 or 24.

(b) A store cell requested for deallocation is not allocated to the TAB.

Message	*Reason and/or action*
(f) MISMATCH	This indicates either: (i) Request header record not input when expected during GBT run. (ii) The request code in the request record does not agree with the request code issued by a bead at the point in the GBT run for which the record was submitted.
(h) MUTILATION OF DRIVER	The contents of HDRTABSTORE have changed between entry to, and exit from a particular bead. This location has therefore been corrupted by the bead.
(j) MUTILATION OF TAB CONTROL INFORMATION	Contents of words 22 to 24 of the TAB (Message, Input/Output and Additional Core area link addresses) have changed between entry to, and exit from, a particular bead. They have therefore been corrupted by the bead, which must be amended. Note that these three words are the only part of the TAB checked by GBT for corruption.
2 GBT OUTPUT FILE EXHAUSTED	The END record on the GBT output file has been found before it was expected. This may happen if either (i) During a run in compare mode, the GBT output and Compare files are found to be identical.

Message		*Reason and/or action*
		(ii) Selection of records by message identification has been specified, and no records with the required identification are present on the GBT output file.
3	INVALID EDIT REC *record image*	An Edit record submitted to Selective Print contains an error. The 80 character image of the record in error is printed as part of the message.
4	INVALID SELECT REC *record image*	Invalid Select records are listed in the same format as invalid Edit records

Appendix 1

Request Codes and parameters

The tables which follow show the format of all requests for which fully specified Driver standards currently exist. The various codes and parameters are shown as they appear in the request area (words 0 to 8) of the TAB.

FUNCTION

Transfer control to another Bead

Relevant additive options: Bit 9 No
Bit 10 No
Bit 11 No

Word 0 Ch.0 = 0
Request Ch.1 = 0
Code Ch.2 = 0
Ch.3 = 0

Word 1

Word 2

Word 3

Word 4

Word 5

Word 6

Word 7

Word 8 Reserved for error reply parameter

FUNCTION

Free a Bead for use by another Tab

Relevant additive options: Bit 9 No
Bit 10 No
Bit 11 No

Word 0 Request Code	Ch.0 = 0
	Ch.1 = 0
	Ch.2 = 0
	Ch.3 = 1
Word 1	Bead Number
Word 2	
Word 3	
Word 4	
Word 5	
Word 6	
Word 7	
Word 8	Reserved for error reply parameter

FUNCTION

Output a message (Message Buffering System)

Relevant additive options: Bit 9 Yes
Bit 10 Yes
Bit 11 Yes

Word 0 Request Code	Ch.0 = 2 Ch.1:- Bits 6 to 8 = 001; Bits 9 to 11, additive options Ch.2 = 1 Ch.3 = 0
Word 1	Device No. (Zero if output is to more than one device.)
Word 2	Length of message in characters
Word 3	Displacement by which to locate message is message area
Word 4	Required setting of HMPUC wd 6 (see *Data Communications and Interrogation* page 206.14)
Word 5	TP number or CCU number
Word 6	Modifier to progress down list in word 7
Word 7	Start address of device list (if output is to more than one device)
Word 8	Reserved for error reply parameter

FUNCTION

Output a message (Scanner only systems)

Relevant additive options: Bit 9 Yes
Bit 10 Yes
Bit 11 Yes

Word 0 Request Code	Ch.0 = 2 Ch.1:- Bits 6 to 8 = 010 Bits 9 to 11, additive options Ch.2 = 1 Ch.3 = 0
Word 1	Required setting of HMPUC Word 0, i.e. line terminal number. See *Data Communications and Interrogation* manual, Chapter 8 page 155
Word 2	Required setting of HMPUC + 1 i.e. 7020 details and length of message in characters. Bit 0 of HMPUC + 1 will be set to 1 by Driver
Word 3	Displacement by which to locate message in message area
Word 4	
Word 5	
Word 6	Modifier to progress down list in Word 7
Word 7	Start address of device list (if output is to more than one device)
Word 8	Reserved for error reply parameter

FUNCTION

Open a TP or CCU (Message Buffering System)

Relevant additive options: Bit 9 Yes
Bit 10 Yes
Bit 11 Yes

Word 0 Request Code	Ch.0 = 2 Ch.1:- Bits 6 to 8 = 001; Bits 9 to 11, additive options Ch.2 = 2 Ch.3 = 0
Word 1	TP or CCU number. (Zero if more than one is to be opened.)
Word 2	
Word 3	
Word 4	
Word 5	
Word 6	Modifier to progress down list in word 7
Word 7	Start address of device list if more than one TP/CCU is to be opened
Word 8	Reserved for error reply parameter

FUNCTION

Close a TP or CCU (Message Buffering System)

Relevant additive options: Bit 9 Yes
Bit 10 Yes
Bit 11 Yes

Word 0 Request Code	Ch.0 = 2 Ch.1:- Bits 6 to 8 = 001; Bits 9 to 11, additive options Ch.2 = 3 Ch.3 = 0
Word 1	TP or CCU number. (Zero if more than one is to be closed.)
Word 2	
Word 3	
Word 4	
Word 5	
Word 6	Modifier to progress down list in word 7
Word 7	Start address of device list (if more than one TP/CCU is to be opened)
Word 8	Reserved for error reply parameter

FUNCTION

Close a device, discarding stored data (Message Buffering System)

Relevant additive options: Bit 9 Yes
Bit 10 Yes
Bit 11 Yes

Word 0 Request Code	Ch.0 = 2 Ch.1:- Bits 6 to 8 = 001; Bits 9 to 11, additive options Ch.2 = 4 Ch.3 = 0
Word 1	Device identifier. (Zero if more than one is to be closed.)
Word 2	
Word 3	
Word 4	
Word 5	
Word 6	Modifier to progress down the table in word 7
Word 7	Start address of device list (if more than one device is to be closed)
Word 8	Reserved for error reply parameter

FUNCTION

Open a device (Message Buffering System)

Relevant additive options: Bit 9 Yes
Bit 10 Yes
Bit 11 Yes

Word 0 Request Code	Ch.0 = 2 Ch.1:- Bits 6 to 8 = 001; Bits 9 to 11, additive options Ch.2 = 4 Ch.3 = 1
Word 1	Device identifier. (Zero if more than one is to be opened.)
Word 2	
Word 3	
Word 4	
Word 5	
Word 6	Modifier to progress down the list in word 7
Word 7	Start address of the device list (if more than one device is to be opened)
Word 8	Reserved for error reply parameter

FUNCTION

Open a device - start for input (Message Buffering System)

Relevant additive options: Bit 9 Yes
Bit 10 Yes
Bit 11 Yes

Word 0 Request Code	Ch.0 = 2 Ch.1:- Bits 6 to 8 = 001; Bits 9 to 11, additive options Ch.2 = 4 Ch.3 = 2
Word 1	Device identifier. (Zero if more than one device is to be opened.)
Word 2	
Word 3	
Word 4	
Word 5	
Word 6	Modifier to progress down the list in word 7
Word 7	Start address of device list (if more than one device is to be opened)
Word 8	Reserved for error reply parameter

FUNCTION

Open a device - start for input of one Message Section (Message Buffering System)

Relevant additive options: Bit 9 Yes
Bit 10 Yes
Bit 11 Yes

Word 0 Request Code	Ch.0 = 2 Ch.1:- Bits 6 to 8 = 001; Bits 9 to 11, additive options Ch.2 = 4 Ch.3 = 3
Word 1	Device identifier. (Zero if more than one device is to be opened.)
Word 2	
Word 3	
Word 4	
Word 5	
Word 6	Modifier to progress down the list in word 7
Word 7	Start address of device list (if more than one device is to be opened)
Word 8	Reserved for error reply parameter

FUNCTION

Close a device - after clearing stored data (Message Buffering System)

Relevant additive options: Bit 9 Yes
Bit 10 Yes
Bit 11 Yes

Word 0	Ch.0 = 2
Request	Ch.1:- Bits 6 to 8 = 001;
Code	Bits 9 to 11, additive options
	Ch.2 = 4
	Ch.3 = 4
Word 1	Device identifier. (Zero if more than one device is to be closed.)
Word 2	
Word 3	
Word 4	
Word 5	
Word 6	Modifier to progress down the list in word 7
Word 7	Start address of device list (if more than one device is to be closed)
Word 8	Reserved for error reply parameter

FUNCTION

Set a device which is "open and started" to be open only (Message Buffering System)

Relevant additive options: Bit 9 Yes
Bit 10 Yes
Bit 11 Yes

Word 0 Request Code	Ch.0 = 2 Ch.1:- Bits 6 to 8 = 001; Bits 9 to 11, additive options Ch.2 = 4 Ch.3 = 5
Word 1	Device identifier. (Zero if more than one device is to be affected.)
Word 2	
Word 3	
Word 4	
Word 5	
Word 6	Modifier to progress down the list in word 7
Word 7	Start address of device list (if more than one device is to be affected
Word 8	Reserved for error reply parameter

FUNCTION

Change the priority of a device (Message Buffering System)

Relevant additive options: Bit 9 Yes
Bit 10 Yes
Bit 11 Yes

Word 0 Request Code	Ch.0 = 2 Ch.1:- Bits 6 to 8 = 001; Bits 9 to 11, additive options Ch.2 = 4 Ch.3 = 6
Word 1	Device identifier. (Zero if more than one device is to be affected.)
Word 2	New priority
Word 3	
Word 4	
Word 5	
Word 6	Modifier to progress down the list in word 7
Word 7	Start address of device list (if more than one device is to be affected)
Word 8	Reserved for error reply parameter

FUNCTION

Change the mode and/or message size of a device (Message Buffering System)

Relevant additive options: Bit 9 Yes
Bit 10 Yes
Bit 11 Yes

Word 0 Request Code	Ch.0 = 2 Ch.1:- Bits 6 to 8 = 001; Bits 9 to 11, additive options Ch.2 = 4 Ch.3 = 7
Word 1	Device identifier. (Zero if more than one device is to be affected.)
Word 2	New Mode/Message size (in count/modifier format)
Word 3	
Word 4	
Word 5	
Word 6	Modifier to progress down the list in word 7
Word 7	Start address of device list (if more than one device is to be affected)
Word 8	Reserved for error reply parameter

FUNCTION

Cause the Housekeeping to cease specifying Restart Points (Message Buffering System)

Relevant additive options: Bit 9 Yes
Bit 10 Yes
Bit 11 Yes

Word 0	Ch.0 = 2
Request	Ch.1:- Bits 6 to 8 = 001;
Code	Bits 9 to 11, additive options
	Ch.2 = 4
	Ch.3 = 8
Word 1	Identifier of the device for which the housekeeping is to cease performing this function. (Zero if more than one device is to be affected.)
Word 2	
Word 3	
Word 4	
Word 5	
Word 6	Modifier to progress down the list in word 7
Word 7	Start address of device list (if more than one device is to be affected)
Word 8	Reserved for error reply parameter

FUNCTION

Cause the Housekeeping to commence specifying Restart Points (Message Buffering System)

Relevant additive options: Bit 9 Yes
Bit 10 Yes
Bit 11 Yes

Word 0 Request Code	Ch.0 = 2 Ch.1:- Bits 6 to 8 = 001; Bits 9 to 11, additive options Ch.2 = 4 Ch.3 = 9
Word 1	Identifier of the device for which the housekeeping is to commence performing this function. (Zero if more than one device is to be affected.)
Word 2	
Word 3	
Word 4	
Word 5	
Word 6	Modifier to progress down the list in word 7
Word 7	Start address of device list (if more than one identifier is to be affected)
Word 8	Reserved for error reply parameter

FUNCTION

Test availability of a (bulk) device (Message Buffering System)

Relevant additive options: Bit 9 Yes
Bit 10 Yes
Bit 11 Yes

Word 0	Ch.0 = 2	
Request	Ch.1:-	Bits 6 to 8 = 001;
Code		Bits 9 to 11, additive options
	Ch.2 = 4	
	Ch.3 = 10	
Word 1	Device identifier. (Zero if more than one device affected)	
Word 2		
Word 3		
Word 4		
Word 5		
Word 6	Modifier to progress down the list in word 7	
Word 7	Start address of device list (if more than one device is affected)	
Word 8	Reserved for error reply parameter	

FUNCTION

Open a line (Scanner Only Systems)

Relevant additive options: Bit 9 Yes
Bit 10 Yes
Bit 11 Yes

Word 0 Request Code	Ch.0 = 2 Ch.1:- Bits 6 to 8 = 010; Bits 9 to 11, additive options Ch.2 = 4 Ch.3 = 1
Word 1	Line number. (Zero if more than one line is to be affected.)
Word 2	0
Word 3	
Word 4	
Word 5	
Word 6	Modifier to progress down the list in word 7
Word 7	Start address of device list (if more than one line is to be affected)
Word 8	Reserved for error reply parameter

FUNCTION

Close a line (Scanner Only Systems)

Relevant additive options: Bit 9 Yes
Bit 10 Yes
Bit 11 Yes

Word 0 Request Code	Ch.0 = 2 Ch.1:- Bits 6 to 8 = 010; Bits 9 to 11, additive options Ch.2 = 4 Ch.3 = 10
Word 1	Line number. (Zero if more than one line is to be affected.)
Word 2	0
Word 3	
Word 4	
Word 5	
Word 6	Modifier to progress down the list in word 7
Word 7	Start address of device list (if more than one line is to be affected)
Word 8	Reserved for error reply parameter

FUNCTION

Give number of transmissions and transmission errors (Character Buffering System)

Relevant additive options: Bit 9 Yes
Bit 10 Yes
Bit 11 Yes

Word 0 Request Code	Ch.0 = 2 Ch.1:- Bits 6 to 8 = 010; Bits 9 to 11, additive options Ch.2 = 4 Ch.3 = 4
Word 1	Line number. (Zero if more than one line is to be affected.)
Word 2	0
Word 3	
Word 4	
Word 5	
Word 6	Modifier to progress down the list in word 7
Word 7	Start address of the device list (if more than one line is to be affected)
Word 8	Reserved for error reply parameter

FUNCTION

Multiplexor - Close Request

Relevant additive options: Bit 9 No
Bit 10 Yes
Bit 11 Yes

Word 0 Ch.0 = 2
Request Ch.1:- Bits 6 to 8 = 010;
Code Bits 9 to 11, additive options
Ch.2 = 3
Ch.3 = 0

Word 1

Word 2

Word 3

Word 4

Word 5

Word 6

Word 7

Word 8 Reserved for error reply parameter

FUNCTION

Form a link (Scanner Only Systems)

Relevant additive options: Bit 9 No
Bit 10 Yes
Bit 11 Yes

Word 0 Request Code	Ch.0 = 2 Ch.1:- Bits 6 to 8 = 010; Bits 9 to 11, additive options Ch.2 = 4 Ch.3 = 3
Word 1	Line number
Word 2	Channel number
Word 3	
Word 4	
Word 5	
Word 6	
Word 7	
Word 8	Reserved for error reply parameter

FUNCTION

Swap Links (Scanner Only Systems)

Relevant additive options: Bit 9 No
Bit 10 Yes
Bit 11 Yes

Word 0 Ch.0 = 2
Request Ch.1:- Bits 6 to 8 = 010;
Code Bits 9 to 11, additive options
Ch.2 = 4
Ch.3 = 2

Word 1 First link

Word 2 Second link

Word 3

Word 4

Word 5

Word 6

Word 7

Word 8 Reserved for error reply parameter

FUNCTION

Allocate store to a TAB

Relevant additive options: Bit 9 No
Bit 10 No
Bit 11 No

Word 0 Request Code	Ch.0 = 2 Ch.1 = 0 Ch.2 = 1 Ch.3 = 0
Word 1	Block number of block containing cells of the required size
Word 2	Destination address of cell (23 or 24)
Word 3	Not used if the request is for one cell; if for two cells, as word 1. Block number in word 3 must be ≥ block number in word 1
Word 4	Not used if the request is for one cell; if for two cells, as word 2. The destination address in this word must not be the same as that in word 2.
Word 5	
Word 6	
Word 7	
Word 8	Reserved for error reply parameter

FUNCTION

Deallocate store from a TAB

Relevant additive options: Bit 9 No
Bit 10 No
Bit 11 No

Word 0	Ch.0 = 3
Request	Ch.1 = 0
Code	Ch.2 = 2
	Ch.3 = 0

Word 1 Block number of block from which the cell to be released was obtained

Word 2 Destination address of cell (23 or 24 as appropriate)

Word 3 Not used if the request is for a single cell; if for two cells, as word 1

Word 4 Not used if the request is for a single cell; if for two cells, as for word 2. The destination address in this word must not be the same as that in word 2.

Word 5

Word 6

Word 7

Word 8 Reserved for error reply parameter

Appendix 2

Driver error codes

On detection of an error during servicing of a request the multithreading Driver will issue one of the following 4-character codes in the form of the error reply parameter placed in word 8 of the TAB.

In each case, the first character (character 0) of the code has the following significance:

Character 0 = 0 Error detected by Request Analyser or Bead Scheduler

Character 0 = 1 Error detected by Peripheral Monitor

Character 0 = 2 Error detected by Communications Monitor

If Character 0 = 2, Character 1 is also significant and contains the package number of the communications housekeeping package in use at the point where the error occurred.

Error codes 0000, and 0001 and 0002, are also issued by the single threading versions of Request Analyser and Bead Scheduler respectively. Error codes issued by user written Peripheral Monitor and Communications Monitor routines should where applicable be the same as those issued by the multithreading versions of these routines.

Error code	*Significance*
0000	Invalid request code [character 0 of word 0 of request code > contents of HDRRACONST]
0001	Bead scheduler request with word 0 of request invalid

Error code	*Significance*
0002	As above, but system in Exception Mode
0003	Invalid bead number [Bead number > contents of HDRBDCONST]
0004	Driver error
0005	Invalid request to store administrator
1006	Invalid file reference number (file reference number > contents of HDRPMCONST) detected on entry to Peripheral Monitor (i.e. within PM Entry routine HDRPMENTRY)
1007	As 1006, but detected in PM Continuation routine HDRPMCONT (i.e. during multiple input or output)
2008	Invalid request. Package number = 0 and bits 10 and 11 of request = 0
2009	Invalid request. Package number non-zero and no corresponding entry in table HDRCMTAB1 (CALL instructions to Communications Monitor interface routines)
200:	Invalid request. Character 2 not in range 1-4
200;	Operator intervention
200<	Invalid MPCRL parameters
200=	Unanticipated systems conditions

Error code	*Significance*
200>	Failure of output request. Transfer completed but number of words as given in word 2 of the tab does not correspond with the number given in word 0 of HMPHR
200?	Failure of output request. Transfer not completed for a reason other than overload (i.e. invalid parameters)
200V	Failure on attempting to obtain input. Bits 1, 4 or 5 of word 0 of HMPHR are set after issue of the MPGET macro
200!	Failure on attempting to obtain input. Word 2 of HMPHR is negative after issue of the MPGET macro