

UNIVERSITY OF EDINBURGH

DEPARTMENT OF COMPUTER SCIENCE

SYSTEM 4-75
EDINBURGH MULTI-ACCESS SYSTEM
REFERENCE MANUAL

SEPTEMBER
1971

EDITED BY
H. WHITFIELD

Work on EMAS began in October 1966 and, until September 1970, was the responsibility of the Edinburgh Multi-Access Project (EMAP) - a joint project between the Department of Computer Science at the University of Edinburgh and International Computers Limited. The then Ministry of Technology paid the University's share of the cost. The present system is the work of D.J. Rees, H. Whitfield and A. S. Wight of the Department of Computer Science (all members of the EMAP team) assisted by S.T. Hayes of the Edinburgh Regional Computing Centre (ERCC) who has had responsibility for the development of demons. Although a great deal of revision has been done in the last year (particularly in connection with the file system) acknowledgement is due to the many members of the EMAP team who contributed ideas and especially to those who wrote the basic device software and the paging and scheduling routines. In particular, R.P. Poole, who wrote the bulk of the paging software and co-ordinated the early testing, deserves special mention. Also deserving credit are P.D. Stephens (ERCC) who wrote the IMPS compiler and G.E. Millard (ERCC) who has made major contributions to the development of sub-systems and who also provided the sub-system (on the non-paged 4-70) on which the early testing was done. J.G. Walker assisted the author in the management of the EMAP team and was co-editor of the four previous issues of this manual. Thanks are also due to Professor S. Michaelson for his enthusiastic and unfailing support over the past five years and to J.G. Burns (ERCC) for his active help and encouragement over the past year.

CONTENTS

A0.	PREFACE	A0 - 1 - 0
A1.	INTRODUCTION	A1 - 1 - 0
A1.1	A PROCESSING UNIT	A1 - 3 - 0
A1.2	VIRTUAL MEMORY	A1 - 4 - 0
A1.3	INPUT-OUTPUT	A1 - 4 - 0
B1.	THE FILE SYSTEM	B1 - 1 - 0
B1.1	FILES AND NAMES	B1 - 1 - 0
B1.2	VIRTUAL MEMORIES	B1 - 2 - 0
B1.3	CONTROLS ON ACCESS TO FILES	B1 - 3 - 0
B1.4	FILE SYSTEM SERVICES	B1 - 4 - 0
B1.5	STORAGE	B1 - 5 - 0
B2.	PROCESS BEHAVIOUR	B2 - 1 - 0
B2.1	STARTING A PROCESS	B2 - 1 - 0
B2.2	SIGNALS	B2 - 1 - 0
B2.3	CONSOLE INTERRUPTS	B2 - 2 - 0
B2.4	REPORTS	B2 - 2 - 0
B2.5	STOPPING A PROCESS	B2 - 2 - 0
C1.	CONSOLE OPERATIONS	C1 - 1 - 0
C1.1	INTRODUCTION	C1 - 1 - 0
C1.2	LOGGING IN	C1 - 1 - 0
C1.3	INTERRUPTIONS	C1 - 3 - 0
C1.4	LOGGING OUT	C1 - 4 - 0
C1.5	STARTING AND STOPPING PROCESSES FROM CONSOLES	C1 - 4 - 0
C2.	INPUT-OUTPUT AND DEMONS	C2 - 1 - 0
C2.1	THE CONSOLE DEMON	C2 - 2 - 0
C2.2	THE SLOW DEVICE DEMONS	C2 - 7 - 0

D1.	METERING AND CHARGING	D1 - 1 - 0
D1.1	METERING	D1 - 1 - 0
D1.2	CHARGING	D1 - 1 - 0
D2.	SCHEDULING STRUCTURE	D2 - 1 - 0
D2.1	INTRODUCTION	D2 - 1 - 0
D2.2	THE QUEUES	D2 - 1 - 0
D2.3	CATEGORIES	D2 - 2 - 0
D2.4	PROMOTION AND RUNNING OF PROCESSES	D2 - 4 - 0
D2.5	LEAVING THE RUN QUEUE	D2 - 5 - 0
D2.6	CORE ALLOCATION	D2 - 6 - 0
D2.7	SUGGESTIONS TO USERS	D2 - 8 - 0
Z1.	CHARACTER SETS AND INTERNAL CODES	Z1 - 1 - 0
Z1.1	EMAS STANDARD CHARACTER SET AND INTERNAL CODE	Z1 - 2 - 0
Z1.2	60 CHARACTER GRAPHIC SUB-SET AND INTERNAL CODES	Z1 - 3 - 0
Z1.3	INPUT - OUTPUT MAPPINGS	Z1 - 4 - 0
Z2.	EXECUTABLE (MACHINE) INSTRUCTIONS - NON-PRIVILEGED	Z2 - 1 - 0
Z2.1	INTRODUCTION	Z2 - 1 - 0
Z2.2	TABLE OF INSTRUCTIONS.	Z2 - 2 - 0
Z2.3	EFFECTS ON CONDITION CODE	Z2 - 7 - 0
Z3.	SYSTEM SERVICES	Z3 - 1 - 0
Z3.1	INTRODUCTION	Z3 - 1 - 0
Z3.2	FILE SERVICES	Z3 - 3 - 0
Z3.3	TIME SERVICES	Z3 -13 - 0
Z3.4	SIGNALS, INTERRUPTS AND STOPPING	Z3 -15 - 0
Z3.5	MISCELLANEOUS SERVICES	Z3 -21 - 0
Z3.6	CONSOLE SERVICES	Z3 -23 - 0
Z3.7	CALLS ON SLOW DEVICE DEMONS	Z3 -29 - 0
Z3.8	PARALLEL EXECUTION OF SERVICES	Z3 -31 - 0

Z4.	REPORTS AND SPECIAL INTERRUPTS	Z4 - 1 - 0
Z4.1	REPORTS	Z4 - 1 - 0
Z4.2	SPECIAL INTERRUPTS	Z4 - 2 - 0
Z5.	DOCUMENT INPUT AND OUTPUT	Z5 - 1 - 0
Z5.1	DOCUMENT CONTROL LANGUAGE(DCL)	Z5 - 1 - 0
Z5.2	MEDIA DEPENDENT VARIATIONS	Z5 - 5 - 0
Z5.3	DISPOSITION OF JOB AND DATA DOCUMENTS	Z5 - 9 - 0

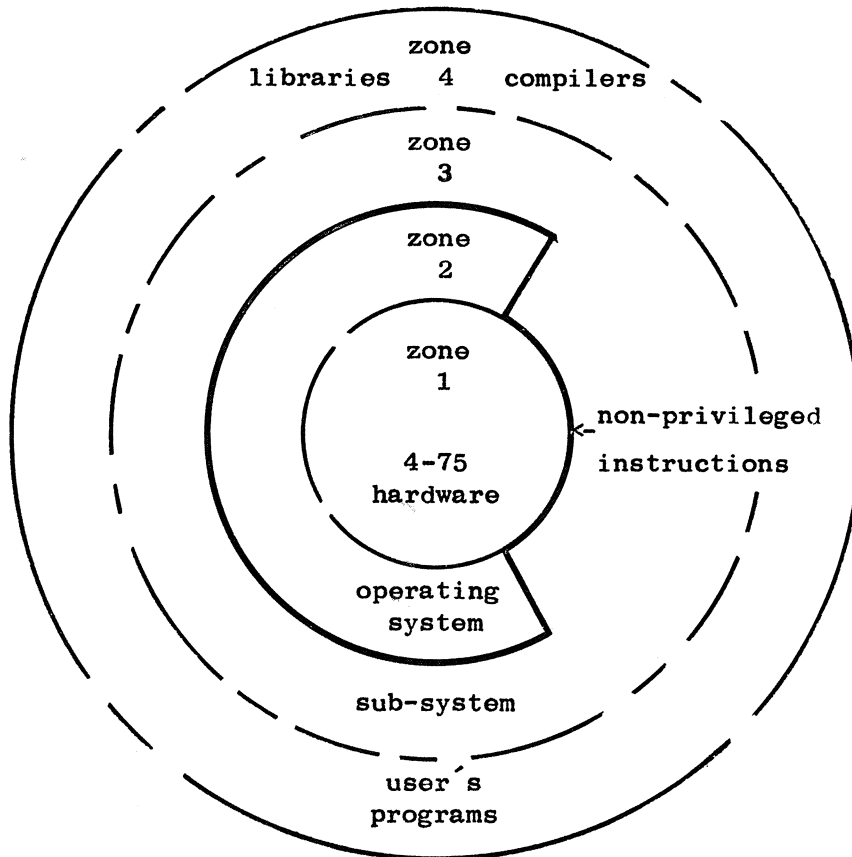
AO. PREFACE

This manual describes the basic facilities of the Edinburgh Multi-Access System (EMAS) written for the ICL System 4-75 computer. It is primarily intended for software programmers and should, together with the EMAS Subsystem Manual, give them all of the information which they require. Operating procedures are described in the EMAS Operators Manual.

A1. INTRODUCTION

The USER is the person (or machine) demanding service.

Service is provided by a complex of hardware and software consisting of the basic hardware of the computer, the operating system, the various subsystems (software utilities for the creation, execution and maintenance of programs), the library programs and the user's own programs.



This manual is concerned with the appearance of the inner two zones of the complex as seen from zone 3. It is zones 1 and 2 which properly constitute the SYSTEM i.e. the part of the whole complex which the ordinary user cannot change.

The execution of the routines associated with a job is referred to as a PROCESS, and the execution of a user's job as a USER PROCESS. A process is owned (and paid for) by the user who initiates it.

The computing resources of the system are shared out amongst the processes which require them so as to give the impression that each process has a processor (albeit somewhat slower than the 4-75 computer) to itself.

In the following description we therefore discuss the interface between zones 1 and 2 and zone 3 in terms of the appearance of such a processor.

A processor consists of a processing unit and an addressing space, areas of which are associated with information held in file storage, some of which is permanent.

A1.1 A PROCESSING UNIT

This is described in two parts.

A1.1.1 THE MILL

This consists of certain registers, whose contents can be changed by program, and the mechanism which effects the changes.

There are 16 General Purpose Registers: R0, ... R15. These are 32-bit registers which are used both as arithmetic and logical accumulators and as index registers in address computations.

There are four 64-bit Floating Point Registers.

There is a condition register CC.

The non-privileged instructions which a user process may use are listed in section Z2.

Effective address computation is performed in the usual System 4 way but an attempt to access addressing space which has not been allocated is trapped (address error).

The SPM instruction can be used in the usual way to inhibit certain fault traps.

The Supervisor Call instruction (SVC) is used to make requests for the system to perform various services (see section Z3).

A1.1.2 THE CONTROL UNIT

A 24-bit register, the Sequence Control Counter (SCC), keeps track of the location of the first byte of the next instruction to be executed. This is incremented as each instruction is fetched to the control unit and can be set by means of branch (jump) instructions in the usual way. The control unit organises the execution cycle in the usual way.

A1.2 VIRTUAL MEMORY

System 4-75 has a 24-bit addressing system and addresses units of one byte (8 bits). This permits an addressing space of 2^{24} (=16,777,216) bytes which is very much larger than the core store of the machine. A processor concerns itself with addresses in this VIRTUAL MEMORY and has no knowledge of actual core addresses.

While in use a virtual memory will address those programs and data files required in connection with a particular job.

A1.3 INPUT-OUTPUT

Input-Output in the usual sense, e.g. line-printer output, is not generally directly controlled by the user process. Instead a file system is provided which enables a user to keep information permanently. Such information is held in FILES which reside in FILE STORAGE.

Before a user can access a file he must set up a virtual memory. In this, byte addresses are associated with that information in file storage to which access is required. The addresses in a virtual memory are allocated in units of 2^{16} (=65,536) bytes, called segments. The least significant 16 bits of an address may be regarded as the byte-address within a segment and the remainder as the segment number. When a process terminates, the association of addresses within its virtual memory with information in file storage is lost. See section B1. for a more complete description.

Actual input-output is performed by processes under the control of the system called DEMONS which are able to transfer information between peripheral devices and files. These processes are more fully described in section C.

B1. THE FILE SYSTEM

B1.1 FILES AND NAMES

B1.1.1 FILES

Information in the system resides in FILE STORAGE and is organised into FILES. Each file is owned by some user and its length is an integral multiple of 2×16 bytes, some of which may be empty. An empty page (or sub-unit of 4096 bytes) does not necessarily have any physical existence.

B1.1.2 USER NAMES

Each user of the system is allocated a unique USER NAME consisting of six characters (not including '.'). The system has a list of all the users who have been authorised to use it.

B1.1.3 FILE NAMES

The system keeps lists of the files belonging to each user. Within a single list a file is referred to by means of a simple name consisting of up to eight characters (not including '.').

A user may refer to his own files by means of this simple name but to refer to another user's files a full file name must be given by prefixing the owner's user name and a '.' to the simple name, e.g. 'MAC007.FRED'

The term FILE NAME is used for a full file name, or for a simple name in those cases where it uniquely specifies a file.

B1.2 VIRTUAL MEMORIES

System 4-75 has a 24 bit addressing system. We therefore set up a mapping between a 2^{24} byte addressing space called a VIRTUAL MEMORY and those files in file storage to which access is required for a given job.

A file is always mapped onto memory starting at a segment boundary and it occupies at least as many contiguous segments as its current length requires.

B1.2.1 CONNECTING AND DISCONNECTING A FILE

The action of associating a file in file storage with one or more segments of a virtual memory is called CONNECTION.

When an attempt is made to connect a file information must be given about how the file is to be accessed, i.e. in read or write mode, to be shared or not. This information is used to check access permission and interlocks with other processes.

DISCONNECTION is brought about by SVC (and on process termination) and frees segment numbers in a virtual memory for association with different files.

B1.3 CONTROLS ON ACCESS TO FILES

Access permission to a file can be granted by its owner to:

- a) himself
- b) selected other users
- c) everyone else.

The permission allows the permitted users access in one or more of the following modes:-

- 1) read shared access
- 2) read + write shared access
- 3) read unshared access
- 4) read + write unshared access

B1.4 FILE SYSTEM SERVICES

Services are provided to enable a user to create, rename and destroy his files, to change the size of his files and to change the modes of access permitted to himself and others. He may also transfer ownership of his files, and interrogate the system about his files and about the access permission he has granted.

Details of these services are given in section Z3.2.

B1.5 STORAGE

The backing store complex of the system is organised into ACTIVE STORAGE and two levels of file storage, IMMEDIATE and ARCHIVE STORAGE.

B1.5.1 ACTIVE STORAGE

Active storage uses the most highly accessible area of backing store, currently two drum units of 2 M-bytes each.

The system uses active storage as a buffer between core and file storage to facilitate the efficient running of active processes.

While a process is AWAKE (i.e. the process is ready to run when core and CPU become available) the pages most actively required by the process are held in active storage. This set of active pages consists of the pages which have been accessed by the process most recently.

B1.5.2 FILE STORAGE

The two levels of file storage (immediate and archive) provide permanent storage for files belonging to the users of the system.

Immediate storage consists of units which are permanently attached to the system and which are not used for active storage, currently one 700 M-byte fixed disc-file. Archive storage uses storage which is not normally attached to the system and which may therefore require operator intervention before it can be used (e.g. magnetic tape).

B2. PROCESS BEHAVIOUR

The normal behaviour of a running process has already been described in section A1. This section deals with the starting and stopping of processes and with abnormal conditions caused by program errors and external events.

B2.1 STARTING A PROCESS

Each user is allowed to start one process which is then identified by his username.

A process is started when a user logs in his first console or when the DEMONS process decides to initiate a background job.

When a process is created the system connects into its virtual memory two SYSTEM FILES, one containing code (the base file), and the other a data area (the base GLA file). The code file is connected in read shared mode and the GLA file is read and write unshared mode. General purpose register R0 is set to the address of the base GLA file and the sequence control counter (SCC) is set to the address of the base file plus sixteen bytes. REPORT 1 (see section Z4) is also queued for the process. Execution then commences.

What happens next is entirely dependent on the action of the program in the base file. A user may specify which files are to be used as the base file and the base GLA file (see section Z3.2). Initially all users have two files specified for them containing components of the standard sub-system.

B2.2 SIGNALS

The recovery mechanism is designed to enable a process to stay in effective control when certain abnormal or external events (SIGNALS) occur. Such events include program error interrupts, time limit checks and certain types of console interrupt.

Recovery is determined by 12 words of recovery information (previously supplied to the system by the user process) which are

used to reset general registers R4-R15 and the program counter (PC- containing the condition code CC, the sequence control counter SCC and the program mask PM) after an error. The system provides services for stacking and unstacking recovery information and for deliberate initiation of the recovery mechanism. When stacking recovery information the user process supplies the address of a 104 byte dump area into which the system later places the values of the process registers at the time of failure. See section Z3.4 for further details.

B2.3 CONSOLE INTERRUPTS

The console interrupt mechanism is described in section C1.3. Console interrupts corresponding to identifiers of more than one character do not cause special system action but are remembered for later interrogation by the user process. This may be likened to the setting of hand-keys on a simple machine. See section Z3.4 for details.

B2.4 REPORTS

In certain cases it is not appropriate to disturb a running process to inform it of some external event e.g. the logging in of a second console. In these cases such REPORTS are queued and can be requested by the process when it wishes. Section Z4.1 lists these reports.

When a process is communicating with several consoles it is convenient for it to be able to issue several I/O requests concurrently. A mechanism is provided to allow this and in this case replies to service calls are returned via the REPORT queue (see section Z3.8).

B2.5 STOPPING A PROCESS

A process may stop itself by issuing the appropriate service call, or it may be stopped by the system if a signal occurs and no recovery information is stacked.

All files are disconnected and any consoles logged in are logged out by the system.

C1. CONSOLE OPERATIONS

C1.1 INTRODUCTION

Console operations are discussed below in relation to a typewriter terminal working in DUPLEX mode. In other words, a typewriter keyboard and printer which are regarded as two independent peripheral devices attached to the central computer with the printer entirely controlled by the computer and not directly connected to the keyboard. When a key is depressed, the computer echos back the character received and thus confirms its correct receipt. In certain circumstances, however, printing does not occur, e.g. when typing a password (see C1.2).

A suitable device for this form of operation is the Model 33 Teletype, but any device presenting the same logical properties (and appropriate electrical properties) could be used. It is also possible to attach devices of a different nature e.g. a satellite computer, but they are not discussed here.

Console input/output is discussed together with other input/output in section C2.

C1.2 LOGGING IN

When a user wishes to begin a session at a console he must first establish a line into the computer system. He does this either by dialling on the public telephone network and using a MODEM or merely by switching on his console if he has a dedicated TELEGRAPH circuit. A modem user cannot establish a line if all such lines at the computer interface are engaged by other users.

Having established a line the LOGGING IN sequence is initiated by the user pressing any key on his console. In response, the message
EDINBURGH MULTI-ACCESS SYSTEM

NAME:

is printed out. The character ':' is used by convention to invite a response from the user. Now the user responds by typing his user name, terminating it with the characters 'CR' (carriage return) and 'LF' (line feed) which are always used to terminate responses, e.g.

NAME:MAC007

The exact form of the name is supplied by the system administrator in order to avoid ambiguity.

Following this, the message

PASSWORD:

is printed out, to which the user should respond with his secret PASSWORD, also supplied by the system administrator. Here, the echoing back of characters typed is suppressed in order to preserve security.

If the user name is recognised as an accredited name for the installation and the password checks against it, the user is given confirmation of being logged in by a message of the form

PROCESS STARTED date time

or

PROCESS RUNNING date time

indicating that a process has been started for him (or is already running) and causing a PROCESS STARTED report (if appropriate) and a CONSOLE LOGIN report to be queued for it (see section B2.4).

He may then continue to use the system and all facilities used are charged to his account.

Should the name be found not to be accredited or the password not to check, the message will be either

INVALID NAME

or

INVALID PASSWORD

as appropriate. The user must then start again from the beginning of the login sequence.

C1.3 INTERRUPTIONS

From time to time, a user may wish to redirect the operation of his process in order either to terminate it completely or to perform some different action. This is achieved by using the control character 'ESC'. Pressing this key causes an interrupt to the system which suspends any input/output operations and prints out on the console

INT:

The system then reads in the characters typed by the user up to 'CR' 'LF'. These characters terminate the interrupt sequence and any input or output operation interrupted is resumed. If no characters were typed in before the 'CR' 'LF', no further action is taken. If, however, there are characters before the 'CR' 'LF' these will constitute an INTERRUPT IDENTIFIER.

Single character identifiers are reserved for special system actions (see section Z4.2), and typically invoke the process error recovery mechanism. Other identifiers set flags which may be interrogated by the process (see section Z3.4).

C1.4 LOGGING OUT

A user process may terminate the association of a console with itself by issuing the LOGOUT service call (see section Z3.6.11). This causes the message

CONSOLE DISCONNECTED date time

to be printed out. The process may still continue if desired.

C1.5 STARTING AND STOPPING PROCESSES FROM CONSOLES

Starting and stopping a process are separate operations from logging in and logging out. However, when a user logs in, a process is started if one owned by him is not already running. If a process stops, the system logs out all the consoles still logged in to it.

C2. INPUT-OUTPUT AND DEMONS

To a process running in its virtual memory, 'input-output' is achieved by transferring information to and from files which have previously been connected to the memory by the normal mechanism (see section B1). The process of transferring information to or from peripheral devices is regarded as being divorced from this, although it may happen concurrently as in the case of consoles. A user will not normally be concerned with peripheral devices himself. Instead, it is the function of the DEMONS to control them and all transfers are achieved by means of requests from a process to one of the demons. The demons currently envisaged are considered below in turn. Other types of device can be used provided that appropriate demons can be written. A set of routines to enable the user process to handle its input/output, in a way consistent with the demon structure, would normally be used in association with the demons.

C2.1 THE CONSOLE DEMON

C2.1.1 GENERAL

The console demon deals with both input and output on a console. Information is transferred between a console and the user process memory when requested by a service call from the user process. Although in the normal case there is just one console and one user process which interact with each user, there may be a number of distinct STREAMS of information both for input and output associated with the process. Facilities are therefore provided by the console demon for multi-stream operation, in such a way as to make up to sixteen input and sixteen output streams associated with up to sixteen different consoles available to the process. The input to or output from a process need not necessarily come from or go to a console even when the process was initiated from a console. The input may already be in a file or the output may need only to be printed out later - perhaps on a line-printer. A console will only be involved when a stream has specifically been COUPLED to the console. Only one input and one output stream may simultaneously be coupled to a console and the same input or output stream may only be coupled to one console at any moment in time. Associations between streams and consoles may be changed dynamically by using the COUPLE and DECOUPLE service calls (see section Z3.6).

C2.1.2 INPUT

For a user process to obtain input from a console, it must first nominate a stream number between 1 and 15 and COUPLE it to the console. The process is only allowed to couple a stream to a console at which the user has successfully completed the login procedure (see section C1.2). This console is identified by the CONSOLE NUMBER passed as a parameter with the LOGIN COMPLETE report (see section Z4). Immediately an input stream has been coupled, the user may start typing his input, i.e. ahead of any requests to GET INPUT made by the process. Normally, however, the user would wait until prompted by the process to type some input. The process PROMPTS by nominating a REQUEST INPUT MESSAGE (see section Z3.6.3). This is a string of characters which will be transmitted to the console when the process requires input and none has yet been typed. For example the message might be

X =

if the process required to read a value for a program variable X. These messages are quite independent of any output stream. Furthermore, they will not be transmitted if any input has already been typed. A process gets input by issuing a GET INPUT service call which nominates the stream from which input is required and the start address of the area in virtual memory to which it is to be transmitted (see section Z3.6.6). Input is transmitted in BLOCKS, rather than character by character. A block consists of either a string of characters terminated by any of the characters 'LF' (line feed), 'EM' (end of medium), 'ETX' (end of text) or 128 characters, which is the maximum number of characters the user/

user may type ahead of get input calls. All completed blocks are transmitted when a get input service call is issued. If there are no complete blocks available at that time, the reply is not sent until there is a block available. This may be avoided by making use of the INPUT AVAILABLE? service call (see section Z3.6.1) which returns the number of characters available. If this number is zero, the process may decide to go off and do something else rather than issue the get input call and be put to sleep.

If the user types 128 characters ahead of get input requests, he is inhibited from typing further characters and '!WAIT' is transmitted to the console. Any characters now typed are treated as interrupt characters and initiate the interrupt sequence (see section C1.3), instead of being appended to the normal input sequence of characters. The user may resume typing when a get input call has been performed and he is notified of this by '!CONTINUE' being transmitted to the console.

C2.1.3 OUTPUT

As in the case of input, for a user process to have output printed on a console, an output stream must first be coupled to the console. The service call COUPLE OUTPUT (see section Z3.6.5) requires a stream number between 0 and 15 and a console number which was passed to the process by a login complete report (see section Z4). An equivalent service call performs the reverse action, that of DECOUPLE OUTPUT (see section Z3.6.10).

Output is transmitted to the console, after it has been coupled, by means of PUT OUTPUT service calls, which nominate the output stream number, an address in virtual memory where the required output characters start, and the number of characters to be transmitted. These are transferred to a buffer area within the demon, from which the actual output operation takes place. If there is space in the buffer for all characters specified, the reply is sent immediately after the service call. If not, the output is transferred to the buffer as and when space becomes available and the reply sent when the last of the output has been transferred. In this way, a whole file of output could be transmitted to the console by issuing one put output call. The amount of space currently available in the output buffer may be determined by the use of the OUTPUT CAPACITY? service call (see section Z3.6.2). As with the equivalent input available? call, the process may avoid being put to sleep by not sending more than the given number of characters to be output. When the number of spaces available reaches 128, all the output previously requested has been transmitted to the console.

In the case of output, unlike input, the demon is not concerned with the construction of suitable blocks. It is left to the output routine issuing the service call to decide whether to output a character at a time as soon as it is received to achieve synchronization between the process and the console, or to output in blocks up to a 'LF' character, say. The criteria for terminating a block may well be the same as for input.

If a further service call is issued while a previous call is still in progress, i.e. before the reply to it has been received, the new call may either be refused or the old call aborted. If the old call was either get input or put output, it will be aborted, an appropriate reply sent and the new call initiated. Otherwise the new call is refused. The exceptions to this are the calls input available?, output capacity? and set input request which are performed without affecting any other call still in progress. The effect of aborting a get input or a put output call is the same as a KILL INPUT or a KILL OUTPUT call respectively (see Z3.6)

C2.2 THE SLOW DEVICE DEMONS

C2.2.1 LINE PRINTER OUTPUT

A user may request that a file be printed on a line-printer by issuing an SVC (see Z3.7.2). The information is passed by the system to one of its subsidiary processes known as DEMONS, which organizes the printing of it as soon as a line-printer becomes available. A heading is printed with the file which includes delivery information specified by the user (see Z3.7.1).

There are two distinct methods of making the file available to DEMONS for printing. The recommended method is to transfer ownership of the file to 'SYSTEM'. DEMONS will rename the file when processing the print request, and eventually destroy it when printing is completed. Clearly, if the file contains information which is of permanent value, then it should be copied before being printed in this way. An alternative method suitable for printing large permanent files, avoiding the overhead of copying, is to give 'SYSTEM' access permission to the file. The file must then not be modified or connected in an incompatible manner until it has actually been printed.

C2.2.2 CARD OUTPUT

As for C2.2.1. See also Z3.7.2 and Z3.7.1.

C2.2.3 PAPER TAPE OUTPUT

As for C2.2.1. See also Z3.7.2 and Z3.7.1.

C2.2.4 CARD READER INPUT

When a user wishes to input cards he does so by submitting properly delimited packs via a card reader. Such a pack consists of a JOB DOCUMENT which may also contain one or more embedded DATA DOCUMENTS. The job document contains a card which specifies the owner's user name and password. Embedded data documents are filed separately, and information giving the file name is placed in the job document by the demon. The demon is responsible for the initiation of BACKGROUND processes and for forwarding job documents to user processes when appropriate. See section Z3.7.3.

Data documents are accepted in 029 and binary codes, and character information is transformed into ISO internal code (see section Z1).

See section Z5 for details of external document control cards and internal file formats.

C2.2.5 PAPER TAPE INPUT

As for C2.2.4 with the obvious exception that data documents are accepted in ISO and binary codes.

D1. METERING AND CHARGING

D1.1 METERING

Metering is performed for two purposes, firstly to enable the services provided by the system to be charged for, and secondly to provide a quantitative record of the behaviour of the system for use in system optimisation.

The following quantities are amongst those metered:-

CPU - central processor compute time in seconds
 PTURNS - number of page transfers to or from core store
 LINES - number of cards/lines of input/output via demons
 FILESPACE - number of PAGES*DAYs of file storage used.

D1.2 CHARGING

The system administrator may define some function of the metered quantities and use this as the system charging function.

Currently the charging system is as follows.

$$\text{CHARGED SECONDS} = \text{CPU} + \frac{\text{PTURNS}}{300} + \frac{\text{LINES}}{200} + \frac{\text{FILE SPACE}}{100}$$

$$\text{UNDISCOUNTED CHARGE} = \pounds \frac{\text{CHARGED SECONDS}}{20}$$

D2. SCHEDULING STRUCTURE.

D2.1. INTRODUCTION.

This section describes those aspects of the current scheduling algorithms which a user might be expected to bear in mind when writing programs for this system.

D2.2 THE QUEUES.

All processes which are active at any instant (i.e., which are actually in control of the CPU, waiting for an allocation of CPU time, waiting for input, or in any other way trying to make progress on some assigned task) are held in one of several queues. The scheduler uses these queues as an aid to allocating the system resources to the various processes demanding service.

D2.2.1. THE RUN QUEUE.

The processes in the run queue are the only ones to which CPU control will be given. In other words, only processes which are in the run queue are allowed to proceed with computation. Generally speaking, the available core store of the machine is shared between the processes in the run queue, so that they may usefully employ the CPU time allocated to them.

A process which is not in the run queue will not in general be kept in core.

Furthermore, the number of processes allowed into the run queue will be determined by core availability.

D2.2.2. THE PRIORITY QUEUES.

Any active processes not in the run queue will be held in one/

one of several priority queues. As processes in the run queue finish their computation, or are removed for other reasons, new processes to take their place are selected out of the priority queues, those processes in the priority 1 queue having a better chance of being selected than those in the priority 2 queue, and so on.

Processes in the priority queues will never be given control of the CPU, but a console I/O operation, say, which was initiated for a process while it was in the run queue, may still be proceeding even though the process has been removed from the run queue back to the priority queues.

D2.3. CATEGORIES.

The priority queue in which a process will be held is one of the items associated with that process's category. Any installation will normally define a whole set of categories, into one of which any active process will fall at any instant. As a process proceeds with its computation, it may in fact be changing from one category to another, and we normally need to know, for any particular process, both the category it is currently in, and the category in which it started. With each separate category there are associated seven parameters, Priority, Residence Time, Process Size and four Next Categories.

D2.3.1. PRIORITY.

All processes currently in a particular category will have the priority, i.e. will be held (when not in the run queue) in the priority queue, associated with that category.

D2.3.2. RESIDENCE TIME.

The residence time of a category is the maximum CPU time that a process in that category is allowed to use before being taken out of the run queue to make room for something else.

D2.3.3. PROCESS SIZE.

Any process in a particular category will, when in the run queue, be allocated core store up to the amount specified by this parameter.

D2.3.4. NEXT CATEGORIES.

Finally, as we remarked above, processes may be changing from one category to another. These parameters determine which category a process will move into next when it leaves the current one.

D2.3.5. The categories are all independent in the sense, for instance, that there is no restriction preventing two different categories having the same associated priority or residence time, although of course any category's specified 'next categories' must actually exist. Moreover there is no restraint insisting, for instance, that large size and low priority must go together, although this may be a sensible way to define the system.

For the system administrator, the manipulation of the parameters of the various categories will provide the easiest means of altering the system's behaviour; for the individual programmer, it will be of crucial importance that his process can assigned to a suitable category.

D2.4. PROMOTION AND RUNNING OF PROCESSES.

When a process leaves the run queue (see D2.5 below) and space becomes free in the core, a new process will usually be promoted from the priority queues to take its place. The following points should be noted.

D2.4.1. High priority processes will be selected for promotion more frequently than low priority processes. Nevertheless, regardless of a process's priority, it will not be possible for a process to have such a low chance of selection that it is NEVER entered. Notice also that it is not the case that all high-priority processes must have been promoted before a low-priority process is given a chance.

D2.4.2. Processes which are currently held up, e.g. which have requested an I/O operation which has not yet terminated, are not eligible for promotion.

D2.4.3. Those processes which have a low residence time, i.e. those which will not stay in the run queue long, have a higher chance of promotion than longer ones. Most interactive processes might be expected to be in the former class.

D2.4.4. Once a process has been promoted to the run queue, it will share the CPU more or less equally with the processes already/

already there. The available CPU time is allocated between the processes in the run queue in short time-slices on a simple round-robin basis. A process which is temporarily held up while, for example, an out-of-core page is brought in, is simply left out of the round-robin.

D2.5. LEAVING THE RUN QUEUE.

After a process has been promoted to the run queue, it is left there, sharing the available CPU time, until one of four things happens.

D2.5.1. The process stops of its own accord.

D2.5.2. The CPU time allocated to the process during this spell in the run queue may reach a total equal to that process's residence time (as determined, of course, by the process's current category). In this case the process is removed from the run queue, its category is changed to the second or third of the four NEXT category parameters, and then the process is put back into the priority queues. The second next category is used when the process has used its full ration of core store, the third when its process size can be reduced. Generally speaking it will be the case that the change of category will have the effect of lowering the process's priority but lengthening its residence time, so that the process will in future have to wait longer for promotion to the run queue, but will stay in the run queue for longer periods once promoted.

D2.5.3. The process may be held up for some reason such as a transfer on a slow I/O device - the console, for instance. In this case again the process will be removed from the run queue, but/

but this time its category will be altered back to the fourth NEXT category before it is put into the priority queues. In general this will mean that the process has returned to a high-priority category, so that when the I/O transfer ends the process will be very quickly selected for the run queue. In this way a rapid response to (for instance) console messages will be maintained.

(Remember that processes are not removed from the run queue when held up for fast peripheral transfers. For instance, the transfer of a page to or from the drum would not cause the process to be put back in the priority queues).

D2.5.4. Finally, if a process tries to use more core than is allocated to it, i.e. more core than processes in its current category are entitled to, then it is removed from the run queue, placed in the first NEXT category and put back into the priority queues. More details of the method of core allocation follow.

D2.6. CORE ALLOCATION.

Each category of processes has a specified process size, which is the maximum amount of core which will be allocated to any process in that category. The following points should be noted.

D2.6.1. Great care is taken by the system that each process is in a category which will entitle it to enough core store to run efficiently. To say that a process needs 10 pages of core to run efficiently is not, of course, the same as saying that the process/

process may only use 10 pages of virtual memory.

Provided that access to virtual memory is arranged in a reasonably orderly fashion, it may only be necessary for a small part of the program and its data to be in core at any one time for high efficiency to be attained.

Occasional switches to another part of virtual memory, for example when a new routine is called, will not cause a great loss of efficiency.

On the other hand, if a process requiring random access to 11 pages of data is run in a category which only entitles it to 10 pages of core, then the process, while it will still run, will do so very inefficiently, and the real time taken will always be very many times higher than the CPU time needed by the process. In this case, the process will normally be placed in a category with a larger process size. The first of the next category parameters determines the next category in this case. The third parameter determines the next category if the system decides to decrease the process size of a process.

D2.6.2. As a process in the run queue proceeds with its computation, it will be allowed to acquire core pages up to the specified allocation. If it attempts to use still more pages, then the process is removed from the run queue and put back into the priority queues to await another turn. However, allowance is made for the fact that a process may need to refer to a page once or twice and then never again (or not for some time). If it appears that a process has not used some particular page for a fair length of time, then that page may be removed from core by the/

the system, thus leaving the process free to acquire a new page without exceeding its allowance.

D2.6.3. In general, when a process is promoted to the run queue from the priority queues, those pages which it was using when it was previously removed will be loaded to core by the supervisor. In this way demand paging will be minimised. However, if the process was removed because it exceeded its process size, then it will normally be allowed to demand page to allow it to change to a new area of virtual memory.

D2.6.4. The overall efficiency of the system will be greatly improved if programmers make use of shared material (e.g. compilers, editing systems). It is therefore in every user's own interest to use shared material where possible.

D2.7. SUGGESTIONS TO USERS.

The following suggestions may help users to achieve high efficiencies and rapid turn-round on the 4-75.

D2.7.1. It should be borne in mind that virtual memory is not in fact a homogeneous random access store.

Programs should be arranged so that all the work involving one part of virtual memory is finished before work involving a different part begins. In particular, it is extremely unlikely that a process will be allowed to occupy a large proportion of the available core store, because the system must serve many users simultaneously.

D2.7.2. Shared material should be used where possible as this will not only increase the speed and efficiency of processes, but will also probably result in a lower cost for computations.

Z1. CHARACTER SETS AND INTERNAL CODES.

The EMAS standard character set (see table 1 below) is based on the 7-bit coded character set for information processing interchange recommended by the International Organization for Standardization (ISO/R646-1967(E)).

Program and data documents can be prepared on a variety of media. The principal external media are cards (IBM 029 code) and paper tape (ISO 7-bit plus even parity paper tape code). In addition, on-line teletype consoles are used.

Normally all character input, however presented, is mapped by the system demons on to the EMAS standard character set and into its internal code.

As the character sets on media preparation and on-line devices are not identical, it is desirable to stay within the compatible subset as far as possible. Table 2 specifies a recommended 60-character subset.

Table 3 gives the mappings for the commonly used characters from the external media to the internal codes and vice versa.

Z1.1. Table 1. EMAS STANDARD CHARACTER SET AND INTERNAL CODE.

								0	0	0	0	1	1	1	1
								0	0	1	1	0	0	1	1
								0	1	0	1	0	1	0	1
b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	COL ROW	0	1	2	3	4	5	6	7
			0	0	0	0	0	NUL	DLE	SP	0	@	P	(4)	p
			0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
			0	0	1	0	2	STX	DC2	"	2	B	R	b	r
			0	0	1	1	3	ETX	DC3	#(1)	3	C	S	c	s
			0	1	0	0	4	EOT	DC4	\$	4	D	T	d	t
			0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
			0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
			0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
			1	0	0	0	8	BS	CAN	(	8	H	X	h	x
			1	0	0	1	9	HT	EM	)	9	I	Y	i	y
			1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
			1	0	1	1	11	VT	ESC	+	;	K	[	k	(4)
			1	1	0	0	12	FF	FS	,	<	L	~(2)	1	
			1	1	0	1	13	CR	GS	-	=	M	]	m	(4)
			1	1	1	0	14	SO	RS	.	>	N	↑(3)	n	(5)
			1	1	1	1	15	SI	US	/	?	O		o	DEL

- (1) The ISO STANDARD specifies # as an alternative to f in this position.
- (2) This variation of the ISO STANDARD is made in accordance with previous practice in Edinburgh.
- (3) ↑ is a typographical variation of the symbol ^.
- (4) These positions have not yet been assigned graphics and should be avoided as future EMAS Standards may assign graphics to them.
- (5) This character (which ISO specifies as ~) is mapped onto ~(92) on input (see note 2 Table 3).

Z1.2. Table 2.

60 CHARACTER GRAPHIC SUB-SET AND INTERNAL CODES.

							0	0	0	0	1	1	1	1	
							0	0	1	1	0	0	1	1	
							0	1	0	1	0	1	0	1	
b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	COL ROW	0	1	2	3	4	5	6	7
			0	0	0	0	0			SP	0	@	P		
			0	0	0	1	1			!	1	A	Q		
			0	0	1	0	2				2	B	R		
			0	0	1	1	3			#(1)	3	C	S		
			0	1	0	0	4			\$	4	D	T		
			0	1	0	1	5			%	5	E	U		
			0	1	1	0	6			&	6	F	V		
			0	1	1	1	7			'	7	G	W		
			1	0	0	0	8			(	8	H	X		
			1	0	0	1	9			)	9	I	Y		
			1	0	1	0	10			*	:	J	Z		
			1	0	1	1	11			+	;	K			
			1	1	0	0	12			,	<	L	~(2)		
			1	1	0	1	13			-	=	M			
			1	1	1	0	14			.	>	N			
			1	1	1	1	15			/	?	O	_		

(1) See note (1) for Table 1.

(2) See notes (2) and (5) for Table 1.

Z1.3. Table 3.

INPUT - OUTPUT MAPPINGS

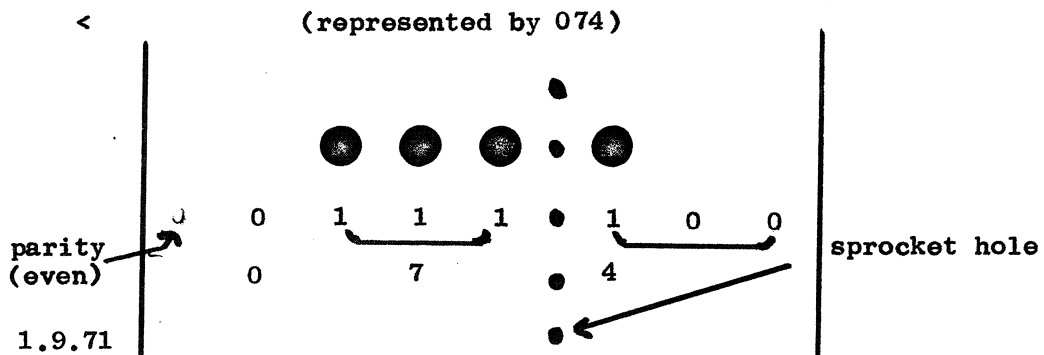
60 CHARACTER SUBSETS

STANDARD GRAPHIC	CARD CODE	PAPER TAPE CODE (OCTAL)	PRINTER	INTERNAL CODE (DECIMAL)
space	blank	040	space	32
!	11-2-8	041	!	33
#(f)	3-8(12-8-2)	043	#(f)	35 (1) and (2)
\$	11-3-8	044	\$	36 (2)
%	0-4-8	045	%	37
&	12	046	&	38
'	5-8	047	'	39
(	12-5-8	050	(	40
)	11-5-8	051	)	41
*	11-4-8	052	*	42
+	12-6-8	053	+	43
,	0-3-8	054	,	44
-	11	055	-	45
.	12-3-8	056	.	46
/	0-1	057	/	47
0	0	060	0	48
1	1	061	1	49
2	2	062	2	50
3	3	063	3	51
4	4	064	4	52
5	5	065	5	53
6	6	066	6	54
7	7	067	7	55
8	8	070	8	56
9	9	071	9	57

STANDARD GRAPHIC	CARD CODE	PAPER TAPE CODE (OCTAL)	PRINTER	INTERNAL CODE (DECIMAL)
:	2-8	072	:	58
;	11-6-8	073	;	59
<	12-4-8	074	<	60
=	6-8	075	=	61
>	0-6-8	076	>	62
?	0-7-8	077	?	63
@	4-8	100	@	64
A	12-1	101	A	65
B	12-2	102	B	66
C	12-3	103	C	67
D	12-4	104	D	68
E	12-5	105	E	69
F	12-6	106	F	70
G	12-7	107	G	71
H	12-8	110	H	72
I	12-9	111	I	73
J	11-1	112	J	74
K	11-2	113	K	75
L	11-3	114	L	76
M	11-4	115	M	77
N	11-5	116	N	78
O	11-6	117	O	79
P	11-7	120	P	80
Q	11-8	121	Q	81
R	11-9	122	R	82
S	0-2	123	S	83

STANDARD GRAPHIC	CARD CODE	PAPER TAPE CODE (OCTAL)	PRINTER	INTERNAL CODE (DECIMAL)
T	0-3	124	T	84
U	0-4	125	U	85
V	0-5	126	V	86
W	0-6	127	W	87
X	0-7	130	X	88
Y	0-8	131	Y	89
Z	0-9	132	Z	90
-	0-5-8	137	-	95
⌈	11-7-8	176	⌈ (3)	92
⌋		134	⌋ (3)	92

- (1) # and ⌈ are equivalent on input. On output the call to the demon can select one or the other.
- (2) IBM 029 (U.K.) code is not identical with IBM 029 (U.S.A.) code. EMAS, like other System 4 regimes, uses the U.S.A. variant. On U.K.-coded equipment, to obtain punching 11-3-8 the key marked ⌈ should be used, and for punching 12-8-2 the key marked \$ should be used.
- (3) The graphic ⌋ is not available on the 4-75 line-printer, or on standard cards, whereas ⌈ is not available on teletype consoles. On input ⌈ is converted to ⌋. On output to the line-printer or cards ⌋ is converted to ⌈.
- (4) Card rows are numbered from the top:
12 11 0 1 2 3 4 5 6 7 8 9
- (5) Example of paper tape punching:



Z2. EXECUTABLE (MACHINE) INSTRUCTIONS - NON-PRIVILEGED.

Z2.1. INTRODUCTION.

The machine instructions for SYSTEM 4-75 can be divided into the following categories:-

- a) Fixed point instructions
- b) Decimal arithmetic instructions
- c) Logical instructions
- d) Branching instructions
- e) Processor state control instructions
- f) Floating point instructions

The instructions in each group are listed below, and described under the following headings:-

Format - The Usercode format of the instruction.

CC - The possible condition code settings and their meanings after the instruction has been executed. The symbols used are explained in Section Z2.3.

Section - Reference to Chapter 6 of the System 4-50/4-70 "Usercode Reference Manual". Each section contains a detailed description of what the action does, the Machine and Usercode formats of the instruction, the interrupt conditions which can occur, and comments or special points relating to the instructions.

Z2.2. TABLE OF INSTRUCTIONS.

FIXED-POINT INSTRUCTIONS.

MNEMONIC	DESCRIPTION	FORMAT	CC	SECTION
A	Add Word	!(RX) R,DXB!	a	6.2.8
AH	Add Halfword	!(RX) R,DXB!	a	6.2.9
AL	Add Logical	!(RX) R,DXB!	h	6.2.10
ALR	Add Logical	!(RR) R,R	h	6.2.10
AR	Add Word	!(RR) R,R	a	6.2.8
C	Compare Word	!(RX) R,DXB!	c	6.2.15
CH	Compare Halfword	!(RX) R,DXB!	c	6.2.14
CR	Compare Word	!(RR) R,R	c	6.2.15
CVB	Convert to Binary	!(RX) R,DXB!	SAME	6.2.19
CVD	Convert to Decimal	!(RX) R,DXB!	SAME	6.2.20
D	Divide	!(RX) R,DXB!	SAME	6.2.18
DR	Divide	!(RR) R,R	SAME	6.2.18
L	Load Word	!(RX) R,DXB!	SAME	6.2.1
LCR	Load Complement	!(RR) R,R	j	6.2.4
LH	Load Halfword	!(RX) R,DXB!	SAME	6.2.2
LM	Load Multiple	!(RS)R,R,DB!	SAME	6.2.7
LNR	Load Negative	!(RR) R,R	e	6.2.6
LPR	Load Positive	!(RR) R,R	k	6.2.5
LR	Load Word	!(RR) R,R	SAME	6.2.1
LTR	Load and Test	!(RR) R,R	b	6.2.3
M	Multiply Word	!(RX) R,DXB!	SAME	6.2.16
MH	Multiply Halfword	!(RX) R,DXB!	SAME	6.2.17
MR	Multiply Word	!(RR) R,R	SAME	6.2.16
S	Subtract Word	!(RX) R,DXB!	a	6.2.11
SH	Subtract Halfword	!(RX) R,DXB!	a	6.2.12
SL	Subtract Logical	!(RX) R,DXB!	i	6.2.13
SLA	Shift Left Single	!(RS) R,DB	a	6.2.24
SLDA	Shift Left Double	!(RS) R,DB	a	6.2.26
SLR	Subtract Logical	!(RR) R,R	i	6.2.13
SR	Subtract Word	!(RR) R,R	a	6.2.11
SRA	Shift Right Single	!(RS) R,DB	b	6.2.25
SRDA	Shift Right Double	!(RS) R,DB	b	6.2.27
ST	Store Word	!(RX) R,DXB!	SAME	6.2.21
STH	Store Halfword	!(RX) R,DXB!	SAME	6.2.22
STM	Store Multiple	!(RS)R,R,DB!	SAME	6.2.23

DECIMAL ARITHMETIC INSTRUCTIONS.

MNEMONIC	DESCRIPTION	FORMAT	CC	SECTION
AP	Add Packed Decimal	1(SS) DLB,DLB	a	6.3.1
CP	Compare Packed Decimal	1(SS) DLB,DLB	c	6.3.4
DP	Divide Packed Decimal	1(SS) DLB,DLB	SAME	6.3.6
MP	Multiply Packed Decimal	1(SS) DLB,DLB	SAME	6.3.5
MVO	Move With Offset	1(SS) DLB,DLB	SAME	6.3.9
PACK	Pack	1(SS) DLB,DLB	SAME	6.3.7
SP	Subtract Packed Decimal	1(SS) DLB,DLB	a	6.3.2
UNPK	Unpack	1(SS) DLB,DLB	SAME	6.3.8
ZAP	Zero and Add Packed	1(SS) DLB,DLB	a	6.3.3

LOGICAL INSTRUCTIONS.

MNEMONIC	DESCRIPTION	FORMAT	CC	SECTION
CL	Compare Logical	(RX) R,DXB	c	6.4.4
CLC	Compare Logical	(SS)DLB,DB	c	6.4.4
CLI	Compare Logical	(SI)DB,I	c	6.4.4
CLR	Compare Logical	(RR)R,R	c	6.4.4
ED	Edit	(SS)DLB,DB	l	6.4.14
EDMK	Edit and Mark	(SS)DLB,DB	l	6.4.15
IC	Insert Character	(RX)R,DXB	SAME	6.4.9
LA	Load Address	(RX)R,DXB	SAME	6.4.11
MVC	Move Characters	(SS)DLB,DB	SAME	6.4.1
MVI	Move Immediate	(SI)DB,I	SAME	6.4.1
MVN	Move Numerics	(SS)DLB,DB	SAME	6.4.2
MVZ	Move Zones	(SS)DLB,DB	SAME	6.4.3
N	AND	(RX)W,DXB	g	6.4.5
NC	AND	(SS)DLB,DB	g	6.4.5
NI	AND	(SI)DB,I	g	6.4.5
NR	AND	(RR)R,R	g	6.4.5
O	OR	(RX)R,DXB	g	6.4.6
OC	OR	(SS)DLB,DB	g	6.4.6
OI	OR	(SI)DB,I	g	6.4.6
OR	OR	(RR)R,R	g	6.4.6
SLDL	Shift Left Double Logical	(RS)R,DB	SAME	6.4.18
SLL	Shift Left Single Logical	(RS)R,DB	SAME	6.4.16
SRDL	Shift Right Double Logical	(RS)R,DB	SAME	6.4.19
SRL	Shift Right Single Logical	(RS)R,DB	SAME	6.4.17
STC	Store Character	(RX)R,DXB	SAME	6.4.10
TM	Test Under Mask	(SI)DB,I	f	6.4.8
TR	Translate	(SS)DLB,DB	SAME	6.4.12
TRT	Translate and Test	(SS)DLB,DB	m	6.4.13
X	Exclusive OR	(RX)R,DXB	g	6.4.7
XC	Exclusive OR	(SS)DLB,DB	g	6.4.7
XI	Exclusive OR	(SI)DB,I	g	6.4.7
XR	Exclusive OR	(RR)R,R	g	6.4.7

BRANCHING INSTRUCTIONS.

MNEMONIC	DESCRIPTION	FORMAT	CC	SECTION
BAL	Branch and Link	(RX) R,DXB	SAME	6.5.2
BALR	Branch and Link Register	(RR) R,R	SAME	6.5.2
BC	Branch on Condition	(RX) M,DXB	SAME	6.5.1
BCR	Branch on Condition Register	(RR) M,R	SAME	6.5.1
BCT	Branch on Count	(RX) R,DXB	SAME	6.5.3
BCTR	Branch on Count Register	(RR) R,R	SAME	6.5.3
BXH	Branch on Index High	(RS)R,R,DB	SAME	6.5.4
BXLE	Branch on Index Low or Equal	(RS)R,R,DB	SAME	6.5.5
EX	Execute	(RX) R,DXB	**	6.5.6

*M : Four-bit mask field.

**Execute Instruction : CC may be set by execution of modified instruction.

PROCESSOR STATE CONTROL INSTRUCTIONS.

MNEMONIC	DESCRIPTION	FORMAT	CC	SECTION
SPM	Set Program Mask	(RR) R	-	6.7.4
SVC	Supervisor Call	(RR) R,R	SAME	6.7.3

FLOATING POINT INSTRUCTIONS.

MNEMONIC	DESCRIPTION	FORMAT	CC	SECTION
AD	Add Normalised (Long)	(RX) R,DXB	n	6.8.5
ADR	Add Normalised (Long)	(RR) R,R	n	6.8.5
AE	Add Normalised (Short)	(RX) R,DXB	n	6.8.5
AER	Add Normalised (Short)	(RR) R,R	n	6.8.5
AU	Add Unnormalised (Short)	(RX) R,DXB	n	6.8.6
AUR	Add Unnormalised (Short)	(RR) R,R	n	6.8.6
AW	Add Unnormalised (Long)	(RX) R,DXB	n	6.8.6
AWR	Add Unnormalised (Long)	(RR) R,R	n	6.8.6
CD	Compare (Long)	(RX) R,DXB	c	6.8.9
CDR	Compare (Long)	(RR) R,R	c	6.8.9
CE	Compare (Short)	(RX) R,DXB	c	6.8.9
CER	Compare (Short)	(RR) R,R	c	6.8.9
DD	Divide (Long)	(RX) R,DXB	SAME	6.8.13
DDR	Divide (Long)	(RR) R,R	SAME	6.8.13
DE	Divide (Short)	(RX) R,DXB	SAME	6.8.13
DER	Divide (Short)	(RR) R,R	SAME	6.8.13
HDR	Halve (Long)	(RR) R,R	SAME	6.8.10
HER	Halve (Short)	(RR) R,R	SAME	6.8.10
LCDR	Load Complement (Long)	(RR) R,R	b	6.8.2
LCER	load Complement (Short)	(RR) R,R	b	6.8.2
LD	Load (Long)	(RX) R,DXB	SAME	6.8.14
LDR	Load (Long)	(RR) R,R	SAME	6.8.14
LE	Load (Short)	(RX) R,DXB	SAME	6.8.14
LER	Load (Short)	(RR) R,R	SAME	6.8.14
LNDR	Load Negative (Long)	(RR) R,R	p	6.8.4
LNER	Load Negative (Short)	(RR) R,R	p	6.8.4
LPDR	Load Positive (Long)	(RR) R,R	b	6.8.3
LPER	Load Positive (Short)	(RR) R,R	b	6.8.3
LTDR	Load and Test (Long)	(RR) R,R	b	6.8.1
LTER	Load and Test (Short)	(RR) R,R	b	6.8.1
MD	Multiply (Long)	(RX) R,DXB	SAME	6.8.12
MDR	Multiply (Long)	(RR) R,R	SAME	6.8.12
ME	Multiply (Short)	(RX) R,DXB	SAME	6.8.12
MER	Multiply (Short)	(RR) R,R	SAME	6.8.12
SD	Subtract Normalised (Long)	(RX) R,DXB	n	6.8.7
SDR	Subtract Normalised (Long)	(RR) R,R	n	6.8.7
SE	Subtract Normalised (Short)	(RX) R,DXB	n	6.8.7
SER	Subtract Normalised (Short)	(RR) R,R	n	6.8.7
STD	Store (Long)	(RX) R,DXB	SAME	6.8.11
STE	Store (Short)	(RX) R,DXB	SAME	6.8.11
SU	Subtract Unnormalised (Short)	(RX) R,DXB	n	6.8.8
SUR	Subtract Unnormalised (Short)	(RR) R,R	n	6.8.8
SW	Subtract Unnormalised (Long)	(RX) R,DXB	n	6.8.8
SWR	Subtract Unnormalised (Long)	(RR) R,R	n	6.8.8

Z2.3. EFFECTS ON CONDITION CODE.

CC VALUE	0	1	2	3
a	!result zero	!result negative	!result positive	!overflow
b	!result zero	!result negative	!result positive	-
c	!operands equal	!first operand	!first operand	-
	!	!low	!high	!
d	!result zero	-	!result positive	!overflow
e	!result zero	!result negative	-	-
f	!all selected	!selected bits	-	!all selected
	!bits zero	!mixture of	!	!bits ones
	!	!zeros and ones	!	!
g	!result pattern	!result pattern	-	-
	!all zeros	!not all zeros	!	!
h	!result zero	!result non-zero	!result zero	!result non-zero
	!(no carry)	!(no carry)	!(carry)	!(carry)
i	-	!result non-zero	!result zero	!result non-zero
	!	!(no carry)	!(carry)	!(carry)
j	!result zero	!result negative	!result positive	!overflow: set
	!	!	!	!only if the
	!	!	!	!contents of the
	!	!	!	!register
	!	!	!	!specified by R1
	!	!	!	!is the maximum
	!	!	!	!negative number
k	!result zero	-	!result positive	!overflow: set
	!	!	!	!only if the
	!	!	!	!contents of the
	!	!	!	!register
	!	!	!	!specified by R2
	!	!	!	!is the maximum
	!	!	!	!negative number
l	!source field	!source field	!source field	-
	!zero	!negative	!positive	!
m	!first operand	!non-zero	!non-zero	-
	!field is	!character	!character	!
	!exhausted	!found in table	!found in table:	!
	!before a non-	!one or more	!no more	!
	!zero character	!characters have	!characters to	!
	!is encountered	!not been	!be translated	!
	!in the table.	!translated	!	!
	!R1,R2 unchanged	!	!	!
n	!result zero	!result negative	!result positive	!exponent overflow
p	!result zero	!result negative	-	-

Z3. SYSTEM SERVICES

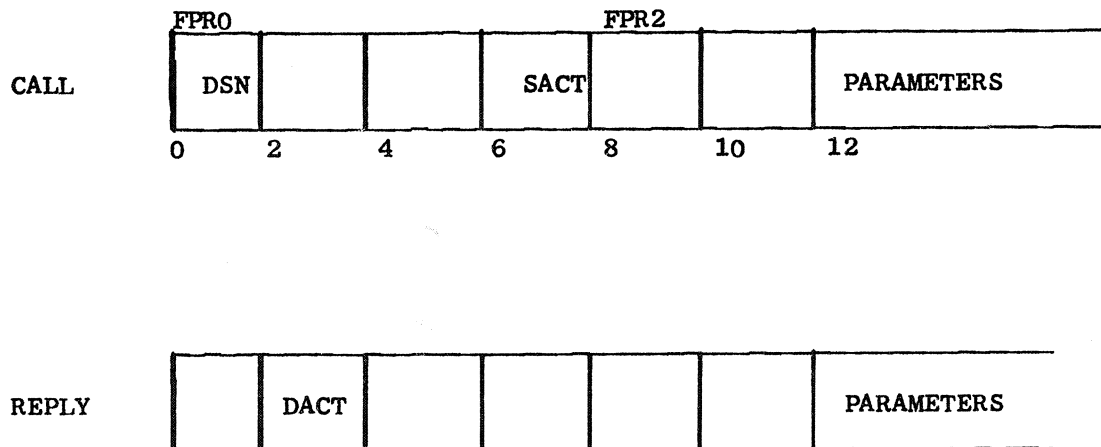
Z3.1 INTRODUCTION

The non-privileged instructions are supplemented by services provided by the system.

These services are invoked by means of SERVICE CALLS. Service calls are performed by placing suitable parameters in the floating-point registers and by issuing the Supervisor Call instruction SVC 254. The floating-point registers contain a REPLY from the system when control is returned to the process.

Z3.1.1 PARAMETER LAYOUT

Parameters are laid out in the floating-point registers as follows:

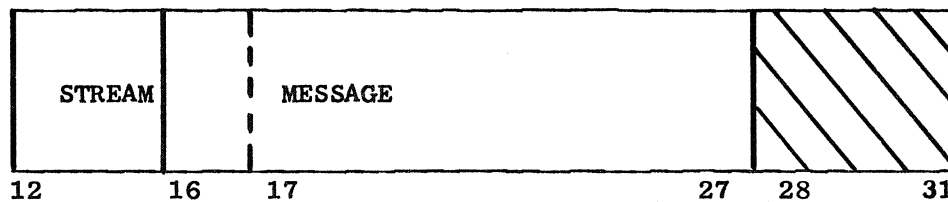


The first twelve bytes of a call or reply are laid out in a standard way. In ordinary service calls only the DSN field is of significance and this is used to select the service required.

The remaining twenty bytes contain parameters peculiar to the particular service. These are described in a conventional manner by giving a format description of as much of the twenty byte record as is required, e.g.

(%INTEGER STREAM, %STRING (11) MESSAGE)

which describes a record laid out as follows:



where STREAM is a 32-bit integer and MESSAGE is a character string of maximum length 11 characters preceeded by a byte which contains the actual string length.

Other types of record element e.g. %SHORTINTEGER, %BYTEINTEGER, etc., occur. Within the record these are always left-justified to the nearest appropriate boundary.

(Those familiar with the IMP language will recognize this as a record format description).

Z3.1.2 PARALLEL EXECUTION OF SERVICE CALLS

When a user process issues an ordinary service call, the process is suspended until the action requested has been performed so that the reply is available.

To enable a process to communicate with several consoles at the same time a facility is provided to enable a process to initiate several service calls without being suspended and to receive the replies later on request (see section Z3.8).

Z3.2 FILE SERVICES

Z3.2.1 SERVICE 160, GET USER/SYSTEM FILE NAME

CALL (%INTEGER NUMBER)

REPLY (%INTEGER FLAG, %STRING (15) NAME)

Provided $0 \leq \text{NUMBER} \leq 7$, FLAG is set to zero and NAME contains information as listed below, otherwise FLAG is set to one.

NUMBER = 0	:	user name of process owner
1	:	base file name
2	:	base GLA file name
3	)	
4	)	available to sub-systems to record
5	)	other system file names
6	)	
7	)	

See section B2.1 for a description of process creation.

Z3.2.2 SERVICE 161, SET SYSTEM FILE NAME

CALL (%INTEGER NUMBER, %STRING (15) NAME)

REPLY (%INTEGER FLAG)

Provided $1 \leq \text{NUMBER} \leq 7$ and NAME is a valid string with length byte ≤ 15 then SYSTEM FILE NAME (NUMBER) is set equal to NAME and FLAG is set to zero, otherwise FLAG is set to one. Numbers of system files as in Z3.2.1 above.

Z3.2.3 SERVICE 162, CREATE FILE

CALL (%STRING (8) NAME, %INTEGER LENGTH)

REPLY (%INTEGER FLAG)

A file with name NAME and length LENGTH pages (which must be a multiple of 16 and not greater than 1024) is created.

NAME must be a simple name.

FLAG = 0 : call successful

1 : invalid parameter

2 : no free file storage left

3 : file already exists

4 : owner's file index full

Z3.2.4 SERVICE 163, DESTROY FILE

CALL (%STRING (8) NAME)

REPLY (%INTEGER FLAG)

The file NAMEd is destroyed.

FLAG = 0 : call successful

1 : invalid file name

2 : file does not exist

3 : file is connected in some virtual memory

Z3.2.5 SERVICE 164, RENAME FILE

CALL (%STRING (8) OLDNAME, NEWNAME)

REPLY (%INTEGER FLAG)

The file OLDNAME is renamed NEWNAME.

FLAG = 0 : call successful

- 1 : invalid parameter
- 2 : old file does not exist
- 3 : file is connected in some virtual memory
- 4 : file with new name already exists

Z3.2.6 SERVICE 165, INCREASE FILE SIZE

CALL (%STRING (8) NAME, %INTEGER INCREASE)
 REPLY (%INTEGER LENGTH, NEWLENGTH)

The NAMED file is increased in size by INCREASE pages (which must be a multiple of 16). The resulting size is returned in NEWLENGTH. The extension is also connected to virtual memory if the file was already connected and if there is a suitable virtual memory gap.

FLAG = 0 : call successful

- 1 : invalid parameter
- 2 : no free file storage left
- 3 : file does not exist
- 4 : resulting file size >1024 pages
- 5 : owner's file index full
- 6 : file connected in another user's virtual memory
- 7 : file connected but insufficient virtual memory gap available.

Z3.2.7 SERVICE 166, TRANSFER OWNERSHIP OF FILE

CALL (%STRING (8) NAME, %STRING (6) NEWOWNER)
 REPLY (%INTEGER FLAG)

The NAMED file is transferred to the NEWOWNER. All access permissions are lost except for X'F' to the new owner.

FLAG = 0 : call successful

1 : invalid parameter

2 : file does not exist

3 : file is connected in some virtual memory

4 : new owner name unknown

5 : new owner already has file of same name

6 : new owner's file index is full

Z3.2.8 SERVICE 167, GET ACCESS PERMISSION

CALL (%STRING (8) NAME, %STRING (6) USER)

REPLY (%INTEGER FLAG, PERMISSION)

This call returns the access PERMISSION granted to USER for the NAMED file (belonging to the owner of the calling process). The value (Rs Ws Ru Wu) is returned as the four least significant bits of PERMISSION.

Rs : read shared access

Ws : read and write shared access

Ru : read unshared access

Wu : read and write unshared access

FLAG = 0 : call successful

1 : invalid parameter

2 : user name unknown

3 : file does not exist

Z3.2.9 SERVICE 168, SET ACCESS PERMISSION

CALL (%STRING (8) NAME, %STRING (6) USER, %INTEGER PERMISSION)
 REPLY (%INTEGER FLAG)

Sets the access PERMISSION (Rs Ws Ru Wu) for the specified USER to the NAMED file (belonging to the owner of the calling process).

FLAG = 0 : call successful
 1 : invalid parameter
 2 : user name unknown
 3 : file does not exist
 4 : owner's file index is full

Z3.2.10 SERVICE 169, CONNECT FILE

CALL (%STRING (15) NAME %SHORTINTEGER SEGMENT, MODE)
 REPLY (%INTEGER FLAG, SIZE)

The file NAMED is connected at the specified SEGMENT (and succeeding ones if necessary) with the specified access mode. The number of segments occupied is returned in SIZE.

The access mode should be set as follows:

MODE = 2 : read unshared
 3 : read and write unshared
 6 : read shared
 7 : read and write shared

When an unshared mode is specified, the file may only be connected in one virtual memory. For shared mode the same file can be simultaneously connected in any number of virtual memories (but only with the same mode in each, read or read and write).

FLAG = 0 : call successful

1 : invalid parameter

2 : owner name unknown

3 : file does not exist or no access permission
is granted

4 : file already connected in another virtual
memory in a conflicting mode

5 : file already connected in this virtual memory

6 : ~~or~~ a segment required for connecting the file is
already in use.

7: top segment > vvm

Z3.2.11 SERVICE 170, DISCONNECT FILE

CALL (%STRING (15) NAME)

REPLY (%INTEGER FLAG, SEGMENT, SIZE)

The file NAMED is disconnected. SEGMENT contains the segment number of where the file started in memory and SIZE contains the number of segments it occupied.

FLAG = 0 : call successful

1 : invalid parameter

2 : file not connected

Z3.2.12 SERVICE 171, CHANGE ACCESS MODE (UNSHARED ONLY)

CALL (%STRING (15) NAME, %INTEGER MODE)

REPLY (%INTEGER FLAG, SEGMENT, SIZE)

A file connected in an unshared mode can have its access mode changed from read to read and write or from read and write to read, without the need for disconnection and reconnection in the alternate mode. The new access MODE which may be 2 or 3 (see Z3.2.10) is

specified for the NAMEd file. SEGMENT and SIZE return the segment number of the start of the file and the number of segments it occupies.

FLAG = 0 : call successful
 1 : invalid parameter
 2 : file not connected
 3 : access mode is 'shared'
 4 : new access mode not permitted
 5 : invalid segment number

Z3.2.13 SERVICE 172, GET FILE INFORMATION

CALL (%STRING (15) NAME)

REPLY (%INTEGER FLAG, SEGMENT, PAGES, MODE, VMS)

Gets information about the NAMEd file.

SEGMENT : segment number at which file is connected in
 this virtual memory, zero if not connected
 PAGES : length of file in pages
 MODE : access mode if connected in any virtual memory
 VMS : number of virtual memories in which connected
 FLAG = 0 : call successful
 1 : invalid parameter
 2 : owner name unknown
 3 : file does not exist or no permission is granted

Z3.2.14 SERVICE 173, GET FILE NAMES

CALL (%INTEGER ADDRESS)

REPLY (%INTEGER FLAG, NUMBER)

ADDRESS should give the base address of a one dimensional array of 20 byte records (in a write permit area of memory) with the format:

(%STRING (15) NAME, %INTEGER PAGES)

On reply the records contain the NAMES of all files belonging to the owner of the calling process and their lengths in PAGES. NUMBER is set to the number of records filled in (i.e. the number of files). At present NUMBER is not greater than 100.

FLAG : 0 : call successful
1 : invalid parameter

Z3.2.15 SERVICE 174, GET VIRTUAL MEMORY MAP

CALL (%INTEGER ADDRESS)

REPLY (%INTEGER FLAG)

ADDRESS should give the base address of a one dimensional array of 20 byte records (in a write-permit area of memory) with the format:

(%STRING (15) NAME, %INTEGER MODE)

On reply the records contain the NAMES of the files connected in this virtual memory and their access MODEs. 96 records are filled in corresponding to segments 32 to 127. If a file occupies more than one segment the name and mode appear in the appropriate number of records.

Z3.3 TIME SERVICES

Z3.3.1 SERVICE 194, DATE AND TIME

CALL

REPLY (%INTEGER DUMMY, %BYTEINTEGERARRAY DATE, TIME (1:8))

Returns the date in DATE as eight characters DD/MM/YY
and the time in TIME as eight characters hh.mm.ss.

Z3.3.2 SERVICE 214, METER

CALL

REPLY (%INTEGER MILLISECS, PAGETURNS, FILE PAGES)

MILLISECS gives the CPU time used by this process. The
CPU time spent doing services for the process as well as CPU
time actually spent in the process is treated as process time.

PAGETURNS gives the number of page transfers performed for the
process.

FILE PAGES gives the total space occupied by files owned by this
user.

Z3.3.3 SERVICE 215, TIME LIMIT

CALL (%INTEGER SECS)

REPLY (%INTEGER FLAG)

The run-time limit of the process is set to SECS (0<SECS<7200)
from the present run-time.

FLAG = 0 : call successful

1 : SECS out of range.

The limit is originally set greater than 7200 seconds. If this limit is violated a signal of weight 132 occurs at the current signal level (see Z3.4). The process is given one second in which to reset the limit, otherwise it is stopped.

Z3.4 SIGNALS, INTERRUPTS AND STOPPING

Z3.4.1 SERVICE 218, SIGNAL

CALL (%INTEGER EP, PARM, EXTRA, ADUMP)

REPLY (%INTEGER A, LEVELS, C, D, FLAG)

Recovery after program errors and other signals is determined by 12 words of recovery information which are used to reset general registers R4-R15 and the PC.

Calls are available to stack and unstack the recovery information and to deliberately initiate the recovery mechanism. The value of EP determines which of the seven versions of the call takes effect.

EP = 0 Stack recovery information

PARM = address of a 48 byte record containing the recovery
information

ADUMP= address of a 104 byte dump area

FLAG = 0 call successful

 = 1 stack full

The recovery information is stacked unless the stack (which has space for 8 levels of stacking) is full. The 104 byte dump area must be in write permit memory and is used by the system when a program error or other signal occurs (see below).

EP = 1 Unstack recovery information

PARM = 0 : unstack one level

≠ 0 : all levels

FLAG = 0 : call successful

= 1 : stack already empty

EP = 2 Signal error of weight PARM at current level. There is no reply as the recovery mechanism is immediately invoked.

EP = 3 Signal error of weight PARM at LEVEL 1.

There is no reply as the recovery mechanism is immediately invoked.

EP = 4 Repeat the last signal at the current level.

There is no reply as the recovery mechanism is immediately invoked.

EP = 5 Change to nominated environment.

PARM = address of 48 byte environment

When processing resumes the process registers are set as follows:

R0 - R3 : Clear

R4 - R15 : set to given environment

FP0 - FP6 : Clear

PC : set equal to R15

EP = 6 Get number of levels stacked.

LEVELS gives number of levels of recovery information currently stacked $0 \leq \text{LEVELS} \leq 8$

When the recovery mechanism is invoked one level of recovery information is unstacked and the registers set as follows:

R0 : Clear
 R1 : Address of dump area
 R2 : Weight of signal
 R3 : Extra information - in the case of deliberate signals (EP=2 or 3) this is set equal to EXTRA.
 R4-R15 : set to recovery information
 FPO-FP6 : Clear
 PC : Set equal to R15

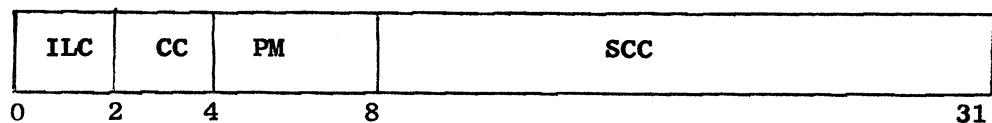
If no levels are stacked the process is stopped.

The dump area nominated in the latest stacking call on SIGNAL (EP=0) is used. The 104 bytes are set to the values of the process registers at the time of the signal which invoked the recovery mechanism. The format of the 104 bytes is as follows:

(%INTEGER WEIGHT, PC, %INTEGERARRAY R(0:15), %REALARRAY FPR(0:7))

WEIGHT is the weight of the signal which caused recovery action

PC has the following format:



where ILC is the INSTRUCTION LENGTH CODE

CC is the CONDITION CODE

PM is the PROGRAM MASK

and SCC is the SEQUENCE CONTROL COUNTER.

The arrays R(0:15) and FPR(0:7) are the General Purpose and Floating Point Registers respectively.

WEIGHTS are as follows:

program errors

- 80 Illegal SVC
- 84 Privileged operation
- 88 Op-code Trap
- 92 Address Error
- 96 Data Error
- 100 Exponent overflow
- 104 Divide error
- 108 Significance error
- 112 Exponent underflow
- 116 Decimal overflow
- 120 Fixed point overflow

other signals

- 128 Operator interrupt (INT:Q and INT:A)
- 132 Local time limit exceeded

Z3.4.2 SERVICE 220, TESTINT

CALL (%INTEGER CNSL, %STRING (15) IDENT)

REPLY (%INTEGER RESULT)

This call is used by the user process to ask if console interrupts have occurred or to clear them. The system remembers a maximum of 8 console interrupts at any one time. Subsequent interrupts are ignored. The console at which an interrupt occurs is also remembered. Duplicate interrupts are ignored.

CLEARING INTERRUPTS

If IDENT = '' (null string) any interrupts associated with console CNSL are cleared. If CNSL = 0 all interrupts are cleared. In either case RESULT = -1.

TESTING INTERRUPTS

If IDENT = '', RESULT indicates whether an interrupt with identifier IDENT has occurred on console CNSL.

If CNSL = 0 all consoles are checked. RESULT = 0 if no such interrupt is recorded otherwise RESULT gives the number of the console. When an interrupt is reported in this way it is cleared.

Z3.4.3 SERVICE 219, STOP PROCESS

CALL

All files are disconnected from the process memory and all consoles are logged out. The process is terminated.

Z3.5 MISCELLANEOUS SERVICES

Z3.5.1 SERVICE 198, CHANGE PASSWORDS

CALL (%STRING (8) FOREGROUND PASS, BACKGROUND PASS)

REPLY (%INTEGER FLAG)

The FOREGROUND PASSWORD, used to accredit a teletype user when he logs on, and the BACKGROUND PASSWORD, used on a job document, are updated. If either password is set to null, then no change is made to that password. For the moment, both passwords must be 4 characters long.

FLAG = 0 : call successful

1 : user not known, or foreground password faulty

2 : background password faulty

Z3.5.2 SERVICE 199, NUMBER OF PROCESSES

CALL

REPLY (%INTEGER NUMBER)

The number of processes currently on the system is returned.

Z3.6 CONSOLE SERVICES

Z3.6.1 SERVICE 201, INPUT AVAILABLE?

CALL (%INTEGER STREAM)

REPLY (%INTEGER FLAG,N)

Are there any complete blocks of input available on console input stream STREAM? (See Z3.6.6., 'GET INPUT', for definition of a 'block' of input). When FLAG returned is zero, N holds the number of characters in the blocks available. ('GET INPUT' withholds its reply if there is no input available). The maximum value of N is 128. After typing this many characters ahead of 'GET INPUT' calls, the console user is told to '!WAIT'. He may continue when a 'GET INPUT' call has been performed and he is notified of this fact by the demon typing '!CONTINUE'.

FLAG = 0 :call performed successfully.

= 1 :STREAM out of range. (See Z3.6.4., 'COUPLE INPUT').

= 2 :no input stream STREAM coupled.

Z3.6.2 SERVICE 202, OUTPUT CAPACITY?

CALL (%INTEGER STREAM)

REPLY (%INTEGER FLAG,N)

Is there spare capacity available in the output buffer for stream STREAM? When FLAG returned is zero, N holds the number of character positions available. (If more than this number of characters is sent by 'PUT OUTPUT', the reply is withheld). It should be noted that 'carriage return' characters will be inserted automatically before each 'line feed' character (unless already present) by the console demon. This should be allowed for in any capacity calculations. The maximum value of N is 128. When this value is reached all output previously requested has been transmitted to the console.

FLAG = 0 :call performed successfully.

= 1 :STREAM out of range. (See Z3.6.5., 'COUPLE OUTPUT').

= 2 :no output stream STREAM coupled.

Z3.6.3 SERVICE 203, SET INPUT REQUEST

CALL (%INTEGER STREAM, %STRING (15) MESSAGE)

REPLY (%INTEGER FLAG)

Set the input request message for stream STREAM to be the string of up to 15 characters MESSAGE. I.e. Upon a 'GET INPUT' call being performed when no input has been typed, this message will be transmitted to the console as a prompt to the user. A string of zero length removes the prompt facility. The initial value of the prompt is set to be ':'. (A length field specification of greater than 15 is regarded as 15). The prompt remains the same until a further call to reset it.

FLAG = 0 :call performed successfully.

= 1 :STREAM out of range. (See Z3.6.4., 'COUPLE INPUT').

= 2 :no input stream STREAM coupled.

Z3.6.4 SERVICE 204, COUPLE INPUT

CALL (%INTEGER STREAM, CONSOLE)

REPLY (%INTEGER FLAG)

An input stream number STREAM ($0 \leq \text{STREAM} \leq 15$) is coupled to console number CONSOLE (the number obtained from a 'login complete' contingency report). This call must be performed before any 'GET INPUT' call. An input stream may only be coupled to one console and a console may only have one input stream coupled to it. There may be both an input and an output stream associated with a particular stream number.

FLAG = 0 :call performed successfully.

= 1 :STREAM or CONSOLE out of range.

= 2 :the required coupling is invalid.

= 3 :there is an operation still in progress on this console.

Z3.6.5 SERVICE 205, COUPLE OUTPUT

CALL (%INTEGER STREAM,CONSOLE)

REPLY (%INTEGER FLAG)

An output stream number STREAM ($0 \leq \text{STREAM} \leq 15$) is coupled to console number CONSOLE (the number obtained from a 'login complete' contingency report). This call must be performed before any 'PUT OUTPUT' call. An output stream may only be coupled to one console and a console may only have one output stream coupled to it. There may be both an output and an input stream associated with a particular stream number.

FLAG = 0 :call performed successfully.

= 1 :STREAM or CONSOLE out of range.

= 2 :the required coupling is invalid.

= 3 :there is an operation still in progress on this console.

Z3.6.6 SERVICE 206, GET INPUT

CALL (%INTEGER STREAM,ADDRESS)

REPLY (%INTEGER FLAG,N)

Transfer all complete blocks of input on stream STREAM to memory from address ADDRESS onwards. On reply, N gives the number of characters transferred (a maximum of 128). A 'block' of input is defined as either a complete buffer (128 characters) or a string of characters terminating with any of the characters LF (line feed), EM (end of medium), ETX (end of text). If no complete block is available, the reply is withheld until such time as there is. If no input has been typed at all, the 'INPUT REQUEST' prompt is transmitted.

FLAG = -1 :call has been aborted (N=0).

= 0 :call performed successfully.

= 1 :STREAM out of range. (See Z3.6.4., 'COUPLE INPUT'), or ADDRESS invalid.

= 2 :input stream STREAM not coupled.

Z3.6.7 SERVICE 207, NOT ASSIGNED.

Z3.6.8 SERVICE 208, PUT OUTPUT.

CALL (%INTEGER STREAM, ADDRESS, N)

REPLY (%INTEGER FLAG, STRM, AD, N)

Transfer N characters from address ADDRESS onwards to output stream STREAM. The reply is withheld if the output buffer capacity (see Z3.6.2.) is less than N and only sent when the last of the output has been transferred to the buffer.

FLAG = -1 : call has been aborted (STRM, AD, and N are filled in with the values necessary for the output to be completed, if required, by a further 'put output' call).

= 0 : call performed successfully.

= 1 : STREAM out of range (see Z3.6.5., 'COUPLE OUTPUT'), ADDRESS invalid, or N <= 0.

= 2 : output stream STREAM not coupled.

Z3.6.9 SERVICE 209, DECOUPLE INPUT

CALL (%INTEGER STREAM)

REPLY (%INTEGER FLAG)

Decouple input stream STREAM from its console. Any input remaining in the buffer is lost. The input stream must be decoupled before a different input stream can be coupled to the same console.

FLAG = 0 : call performed successfully.

= 1 : STREAM out of range. (See Z3.6.4., 'COUPLE INPUT').

= 2 : input stream STREAM not coupled.

Z3.6.10 SERVICE 210, DECOUPLE OUTPUT

CALL (%INTEGER STREAM)

REPLY (%INTEGER FLAG)

Decouple output stream STREAM from its console. All output in the buffer is transferred before the reply is sent. The output stream must be decoupled before a different output stream can be coupled to the same console.

FLAG = 0 :call performed successfully.

= 1 :STREAM out of range. (See Z3.6.5., 'COUPLE OUTPUT').

= 2 :output stream STREAM not coupled.

Z3.6.11 SERVICE 211, LOGOUT

CALL (%INTEGER CONSOLE)

REPLY (%INTEGER FLAG)

Logout console number CONSOLE (the same console number used in 'COUPLE INPUT' and 'COUPLE OUTPUT' obtained from the 'login complete' contingency report). The process is not stopped, but may continue in background mode if required. Input and output streams must be decoupled prior to issuing this call.

FLAG = 0 :call performed successfully.

= 1 :CONSOLE out of range.

= 2 :process is not using console CONSOLE or an input or output stream still coupled.

Z3.6.12 SERVICE 212, KILL INPUT

CALL (%INTEGER STREAM)

REPLY (%INTEGER FLAG)

The 'get input' call on stream STREAM is aborted. An appropriate reply is sent (FLAG = -1).

FLAG = 0 :call performed successfully.

= 1 :STREAM out of range.

= 2 :no input stream STREAM coupled.

= 3 :no 'get input' in progress on this stream.

Z3.6.13 SERVICE 213, KILL OUTPUT

CALL (%INTEGER STREAM)

REPLY (%INTEGER FLAG)

The 'put output' call on stream STREAM is aborted and the output stopped at whatever stage it has reached. An appropriate reply is sent (FLAG = -1).

FLAG = 0 :call performed successfully.

= 1 :STREAM out of range.

= 2 :no output stream STREAM coupled.

= 3 :no 'put output' in progress on this stream.

Z3.7 CALLS ON SLOW DEVICE DEMONS

Z3.7.1 SERVICE 195, UPDATE DELIVERY INFORMATION

CALL (%STRING (19) NEW DELIVERY INFO)

REPLY (%INTEGER FLAG)

The delivery information for output documents is updated. It is used for all following output requests, but does not effect the delivery information for output already requested.

Delivery information may not be null.

Values of FLAG?

Z3.7.2 SERVICE 196, DOCUMENT OUTPUT

CALL (%STRING (15) FILE, %SHORTINTEGER OPTIONS, DEVICE)

REPLY (%INTEGER FLAG)

The file specified is printed or punched as soon as possible.

The device on which output is required is specified by DEVICE.

DEVICE=1 Paper Tape Punch
 OPTIONS=0 ISO MODE
 OPTIONS 1 BINARY MODE

DEVICE=3 Line Printer
 OPTIONS=0 ISO MODE prints internal code 35 as #
 OPTIONS=2 ISO MODE prints internal code 35 as £

DEVICE=5 Card Punch
 OPTIONS=0 ISO MODE punches internal code 35 as # (3-8)
 OPTIONS=1 BINARY MODE
 OPTIONS=2 ISO MODE punches internal code 35 as £ (12-8-2)

The file should be either owned by or accessible to 'SYSTEM'. If it is owned by 'SYSTEM', it will be renamed as the request is processed, and destroyed when printing is finished. For information on output OPTIONS see section Z5.2.

FLAG/

FLAG=0 : call successful

1 : invalid parameter

2 : output list full

3 : faulty device code

4 : user not known

5 : file does not exist, or no access permission is granted

6 : file too big

Z3.7.3 SERVICE 197, JOB DOCUMENT ENQUIRY

CALL (%STRING (8) JOBNAME)

REPLY (%INTEGER FLAG, %STRING (8) JDOC)

This call is used to enquire of the DEMONS whether or not a Job Document is held under the given JOBNAME.

JOBNAME has the form

<USER NAME> <JOB DESCRIMINATOR>

If a document is held, FLAG=0 and JDOC is the filename of the Job Document.

If a file is not held, then FLAG=1. Only the owner of a document will be told of its existence.

A successful enquiry for a document results in the deletion of this document from the input list.

In the case of duplicate documents with the same JOBNAME, JDOC is the filename of the first document held.

If the enquiry is made with JOBNAME null, then the first document belonging to the user is returned.

Z3.8 PARALLEL EXECUTION OF SERVICES

The following services are provided to enable the user process to initiate service requests in parallel and to receive the replies when available.

Z3.8.1 SERVICE 216, PON - REQUEST A CALL

CALL (%INTEGER ADDRESS)

REPLY

ADDRESS is the address of a 32-byte record.

The effect of PON is to request the system to perform the service specified in the 32-byte record. This is laid out in memory in the same way as an ordinary call is laid out in the floating point registers. Processing resumes as soon as the PON is complete - the reply to the service call in the 32-byte record is not returned even if it is available. Instead it is queued, when available, on the report queue.

To enable the replies to be differentiated the calling process should set the SACT field of the 32-byte record to some non-negative value less than 128. This value is returned in the DACT field of the reply. See section Z3.1.1 for the layout of the record. See also section Z4.1 for further restrictions on the values of SACT.

Only certain requests may be made in this way, where the reply could be delayed pending user action at a terminal. Services 201-206 and 208-213 may be sent in this way. An attempt to PON any other service request results in an INVALID SVC signal at the current signal level. The report queue has space for 32 replies or reports. If the queue overflows the process is stopped.

Z3.8.2 SERVICE 217, POFF - REQUEST A REPLY OR REPORT

CALL (%INTEGER ADDRESS)

REPLY

ADDRESS must be the address of a 32-byte record in a write permit area of memory. The process is suspended until a reply or report is available on the report queue. When processing resumes the 32-byte record contains the reply or report.

Z4. REPORTS AND SPECIAL INTERRUPTS

Z4.1 REPORTS

In addition to replies to service requests issued via PON, the following reports are queued on the report queue. The format of a report is the same as that of a reply. The number of the report is given in the DACT field. To avoid confusion, when using PON, SACT should not be set to a valid report number.

Z4.1.1 REPORT 1, PROCESS STARTED

REPORT (%INTEGER REASON, GLABASE, FREESEG)

This report is queued to provide the sub-system with certain essential information.

REASON = 0 : process has been started by a console logging in
 1 : process has been started by DEMONS
GLABASE : address of base GLA file
FREESEG : number of lowest free segment

Z4.1.2 REPORT 6, LOGIN COMPLETE

REPORT (%INTEGER CONSOLE)

This report is queued to indicate that console CONSOLE has logged in to this process.

Z4.2 SPECIAL INTERRUPTS

See section C1.3 for an explanation of the console interrupt mechanism. Interrupts with identifiers of more than one character set flags which can be interrogated by the TESTINT call (see section Z3.4.2). Some single character interrupts cause special actions which are described below. All other single character interrupts are ignored.

Z4.2.1 INT:A

All console I/O is aborted and an OPERATOR TERMINATION signal occurs at level 1.

Z4.2.2 INT:Q

All console I/O is aborted and an OPERATOR TERMINATION signal occurs at the current signal level.

Z5. DOCUMENT INPUT AND OUTPUT

Z5.1 DOCUMENT CONTROL LANGUAGE(DCL)

For a general description of an input DEMON see section C2.2.4.

Four types of DCL statement are recognised by the input DEMONS:

- (1) JOB statement
- (2) DATA Document statement
- (3) Data Document delimiter statement
- (4) Job delimiter statement

Z5.1.1 JOB STATEMENT

This statement marks the beginning of a new job document and, by implication, the end of any improperly terminated previous document.

It takes the form

```
//<user name><job discriminator> JOB (project>,<password>,  
      <resource limit list>,<title>
```

where

<user name>	is the six-character user name
<job discriminator>	is an optional one or two-character field
<project>	is a project number, up to four digits
<password>	is a four character password
<resource limit list>	is a list of integers separated by commas
<title>	is any text

Z5.1.2 DATA DOCUMENT STATEMENT

This statement marks the beginning of an embedded data document, and, where appropriate, marks the end of any previous data document. It takes the form

```
//<ddname> DD <term>,<options list>
```

where

<ddname>	is the name identifying this data document statement
<options list>	is an optional field specifying the transformations to be performed upon the data document before it is filed (see section Z5.2).
<term> = *	means that the next statement with // at the beginning of a line is the terminator.
<term> = DATA	means that the next statement with /* at the beginning of a line is the terminator.
<term> = BINARY	means that the terminator is to be the same pattern as that on the line following this statement.

Z5.1.3 DATA DELIMITER STATEMENT

This statement marks the end of a data document. There are three forms of termination statement corresponding to the three cases of <term> in the data document statement (see above).

In the case <term> = *, the statement beginning with // is not regarded as part of the data document.

In the case <term> = DATA, only a statement of the form
/*

on a line by itself causes termination. The termination statement is not regarded as part of the data document nor is it included in the job document

In the case <term> = BINARY, neither the line specifying the terminator nor the terminating line is regarded as part of the data document nor is it included in the job document.

Z5.1.4 JOB DELIMITER STATEMENT

This statement is the correct way to terminate a job document. It takes the form

//

on a line by itself. Note, however, that this statement will usually have no effect in DATA or BINARY termination modes. It is never included in the job document.

Z5.2 MEDIA DEPENDENT VARIATIONS

The DEMONS provide facilities appropriate to the particular properties of each medium. In addition, there are variations in the options available.

Z5.2.1 CARD READER INPUT

In normal character mode, cards are read and converted to ISO code. Cards containing columns which do not correspond to legal EBCDIC codes cannot be read in this mode, and are rejected. Columns corresponding to EBCDIC codes with no ISO equivalent are translated into the SUB character (X'1A').

Within the job document, columns 73-80 are discarded and trailing spaces are deleted. No double quote deletion is done.

Within data documents introduced by * or DATA the reading mode depends on the options specified. The default is to ignore the identifier field (columns 73-80), do double quote deletion, replacing deleted characters by DEL(127), and to remove trailing spaces. The information is then packed in free format separated by LF(NL) characters.

The following options are valid:

IDENT	Do not ignore columns 73-80
QUOTES	Do not do double quote deletion
TRAIL	Do not delete trailing spaces

Data read in BINARY mode is read as follows. Each card image consists of 80 short integers. The high order bits of each short integer are set to zero, and bits 4-15 contain card rows 12, 11, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. The card images are packed in fixed format, and no LF(NL) characters are inserted.

Z5.2.2 PAPER TAPE READER INPUT

In the normal character mode, null characters and deletes are ignored, and parity errors are transformed into X'1A'. Spaces and carriage returns preceeding line feed are ignored.

In binary mode, there is no parity checking, translation or deletion. The reader hardware ignores runout at the beginning of paper tapes, but not at the end. The termination sequence is defined as LF<PATTERN>LF where <PATTERN> is that on the line following the relevant

```
//ddname> DD BINARY
```

statement. Neither the initial <pattern>LF nor the terminal LF<PATTERN>LF appears in the data document. <PATTERN> must be less than 254 bytes.

Z5.2.3 LINE PRINTER OUTPUT

The output file consists of lines of (ISO) text in free format, separated by one of the following control characters.

LF (NL) = 10

CR = 13 : specifies overprinting

FF = 12 : throws a new page

If a line exceeds 132 characters in length, then newlines are inserted into the output. CR is ignored immediately following any control character. If CR is immediately followed by LF, then the CR is ignored. FF as a terminator has the same effect as LF followed by FF.

Z5.2.4 CARD PUNCH OUTPUT

In character (ISO MODE) output, only LF(NL) is regarded as a terminator. Lines longer than 80 characters are continued on the next card. Line shorter than 80 characters are padded with spaces. The LF(NL) character is not punched.

CR and LF are ignored after LF(NL)

CR followed by LF(NL) is treated as LF(NL)

For binary output, the internal file format is as defined in section Z5.2.1.

Z5.2.5 PAPER TAPE PUNCH OUTPUT

Character (ISO MODE) Mode:

Only LF(NL) is recognised as a terminator, and is replaced by a CR/LF pair. A CR/LF pair is treated as a LF. CR after LF is ignored.

Only one CR is generated before multiple LF's.

There is no maximum line length. Even parity tape is generated. Codes over 127 are converted to X'1A'.

Binary Mode:

The file is punched as the sequence of bytes given. All 8 bits are punched.

Z5 - 8 - 0

S.R.M. 1.9.71

Z5.3. DISPOSITION OF JOB AND DATA DOCUMENTS

Z5.3.1 DATA DOCUMENTS

Data documents are filed separately from the job documents in which they are embedded. At the end of reading a job document, the job and data files are transferred to the owner of the document, as specified in the JOB card. The names of the files used for the data documents are recorded in the corresponding job document, by modifying the DD statement introducing the data document.

The modified DD statement takes the following form:

```
//<ddname> DD <term>,DSN=FILENAME
```

Z5.3.2 JOB DOCUMENTS

When a DEMON starts to read a job document, it picks out the user name and password from the JOB statement. These parameters are checked to ensure that the user is accredited. If not, the job document is discarded.

When a complete job document has been read, ownership of the files used for the job document and its embedded data documents is transferred to the user defined by the job card. Information about the job document is kept until the owner enquires about it. When enquiring, he may give either the full name of a particular document (user name + job discriminator), or a null name. In the latter case, information about the first document held is passed to him.

If the password in the JOB statement is missing, then the DEMON assumes that the document is to be processed by the BATCH system. The files are transferred to the batch system owner, and only the batch system may enquire about these documents.

It is the responsibility of DEMONS to start processes, as system loading permits, in order that they may enquire about and process job documents.

