RCSL : 44 - RT - 759
Date : Rev. September 1974
Authors : Torkild Glaven / AP Ravn

MUS - SYSTEM

INTRODUCTION

(part one of two)

Keywords: Multiprogrammering, monitor, devicehandling, i/o-utility, record i/o, operator communication, operating system.

Abstract: This Manual is intended as an introduction guide to the Multiprogrammering Utility System.
If actual programming is to be performed, MUS programmers Guide should be consulted to get formats and examples of use.

This manual obsoletes RCSL 44 - RT - 614, April 1973.

CONTENTS PAGE

SYSTEM OVERVIEW

The multiprogramming Utility System for DGC NOVA line of computers has the aim

- to implement parallel processing including interprocess comunication and interrupt processing.
- give a strong framework for i/o processing, both on character level and on record oriented level.
- support the user in the running of the system, which includes easy operator communication, and a basic operating system that takes care of the creation, removal of processes and loading or delection of programs to core.

These goals have been reached by creation of the following software modules:

1.1 A multiprogramming monitor (system supervisor), the design of which rests heavily on the proven design of the RC 4000 multiprogramming system.

1.2 Driver programs for common devices. These lay down the rules which are to be followed in coding drivers for new devices. These rules are purely a matter of overall cleanliness, as no real destinction is made between driver programs and ordinary programs.

1.3 Reentrant i/o procedures designed around the zone concept of RC 4000, which has shown itself to be a clean and tidy way to describe device peculiarities, buffering, record formatting and - packing involved in any i/o activity.

1.4 A basic operating system, which caters for program load and deletion, process creation and removal and start or stop of existing processes. This operating system can recieve commands from the human operator or from an external device.

SYSTEM CONFIGURATION

The different modules have been fitted together in a manner, which should give as few logical dependencies as possible and especially eliminate non-hieracial interfaces.

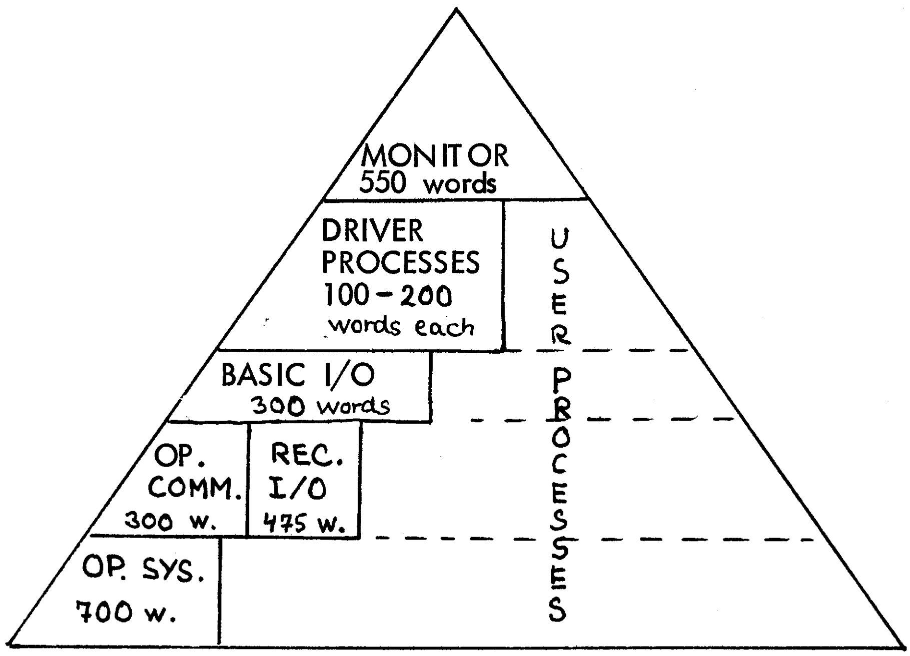
Notice that the following versions of the system may be used as subsystems.

A: Monitor alone

B: Monitor and driver processes

C: B and operator communication and basic i/o procedures

D: and/or record handling procedures.



NOTATION AND TERMINOLOGY

address integer	An address may be a word address, which is a 15 bit unsigned integer, corresponding to a physical address in core store. Or it may be a byteaddress, which is a word address left shifted one and with a one added in bit 15 if the byte addressed within the word is to the right.
bit	A computer word consists of 16 bits, numbered from left to right: B0, B1, B2, ... B15.
byte	A computer word is regarded as two 8 bit bytes. The left one bit0 to bit7 has a even address and the right one bit8 to bit15 an odd address.
character	A character is a byte. There exists no common alphabet within the system; thus there can be no graphic meaning of a byte value.
text	A text is a sequence of characters. Starting at a byte address and containing in a left to right packing. A text is terminated by a Nullcharacter with value zero.
descriptor	A collection of information, which describes an object, is called a descriptor. Descriptors are found as part of items and as part of zones.
item	An item is a core area, which is headed by a descriptor, the first part of which has a standard layout. This ensures that an item always may be in some chain and possibly also in a queue.The first words of an item contains the fields:

next:	next item in a queue
prev:	previous item in a queue
chain:	next item in a chain
size:	the size of the core area of item
name:	(3 words) A text identifying the item.

field	A field is a displacement, which identifies a piece of information within a descriptor. Fields are predefined in the system assembler.

Notation for a field f of a descriptor d is:

f.d

Fields may be used as displacements in assembler code. For example: if accumulator ac2 contains the address of descriptor d, the contents of field f of d may be loaded to ac0 by the instruction

```
lda    0,   f,2   ;   ac0: = f.d
```

chain — (linked linear list). A chain consists of a chain head and a number of chain elements. The head and each element point at the next item in the chain, the last element equals zero.
For example:

chain.head: first   chain.first: last   chain.last: 0

When the chain is empty chain.head equals zero.

queue — (doubly linked cyclical linear list). A queue consists of one or more queue elements. One of the elements is the queue head. A queue element consists of two consecutive words pointing at the next element in the queue and the previous element in the queue respectively.
For example:

next.head: first   next.first: last   next.last: head
prev.head: last   prev.first: head   prev.last: first

When a queue is empty the head points at itself. When an element is not in a queue it normally points at itself.

length — The term length is used to express the number of bytes contained in some core area.

size — The term size is used to express the number of words contained in some core area.

function — A function is a monitor routine executed in disabled mode. Call of a function is executed by writing its name (the linkage is defined in the system assembler) E.g. Function send message (address, name_address, buf)

	call:	return:
ac0		unchanged
ac1	address	address
ac2	name address	buf
ac3	link	cur

A call is coded as:

```
    lda 1,      words       ; ac1: = address of message
    lda 2,      name, 2     ; ac2: = name of item
    sendmessage
```

procedure

A procedure is a system routine executed in enabled mode.
Call as an example for functions.
A procedure may also be called with preloaded link register.
E.g. the following routine may be used to fetch consecutive bytes from an area:

fetchbyte:

```
    lda 1       abyte       ; ac1: = byteaddr;
    isz         abyte       ; increment (byteaddr);
    dsz         count       ; if decrement (count) ≠ 0 then
    .getbyte                ;    begin getbyte (byte,byteaddr);
    jmp         + 1,3       ;              return to (link+0)
                            ;    end;
                            ; return to (link+1);
```

MONITOR

The primary purpose of the monitor is to implement multiprogramming, that is simulation of parallel execution of several active programs (processes) on a single physical processor.
In order to do this, the normal operation is interrupted at regular intervals by a real time clock device (rtc). When such an interrupt occurs the monitor gains control of the processor, and is able to determine which process is to get the next slice of time for instruction execution.

As interrupts from devices are intercepted by the monitor, it also includes interrupt handling functions. Use of this facility give processes the ability to synchronize with devices. Futhermore this waitinterrupt function is extended with a software timer, if the device does not interrupt within a given number of rtc periods.

The need for synchronization also exists within the group of processes, and the monitor implements this as a facility to exchange fixed amounts of information between processes in such a way that only one process at a time accesses the information.

All information about a process, which is needed by the monitor is collected in a process descriptor .

	monitor usage.
description, links to chain and queues	
process name	identification
process state	dispatch information
process delay period	
priority of process	
saved registers, program counter, and carry	
communication description	interprocess communication
program information	start, stop process.
optional words	not used by monitor

fig. 2.1 Process Descriptor.

All process descriptors are linked together in a process chain. Those processes which are running (i.e. competing for the time slices) are also linked into the running queue. The available slice is always given to the process at the head of running queue. Insertions into running queue is done in order of priority (a positive integer). Processes of equal priority are inserted in order of insertion.

If a process is not running, it may be waiting for an event (synchronizing with another process), waiting for an interrupt, or stopped.
In all cases of waiting it may be linked to delay queue, if it has specified a number of delay periods it wants to wait. In case nothing happens, it will be set running, when this number of timer periods have elapsed. Waiting for interrupt it is also attached to a device table which determines which device number it is waiting for.

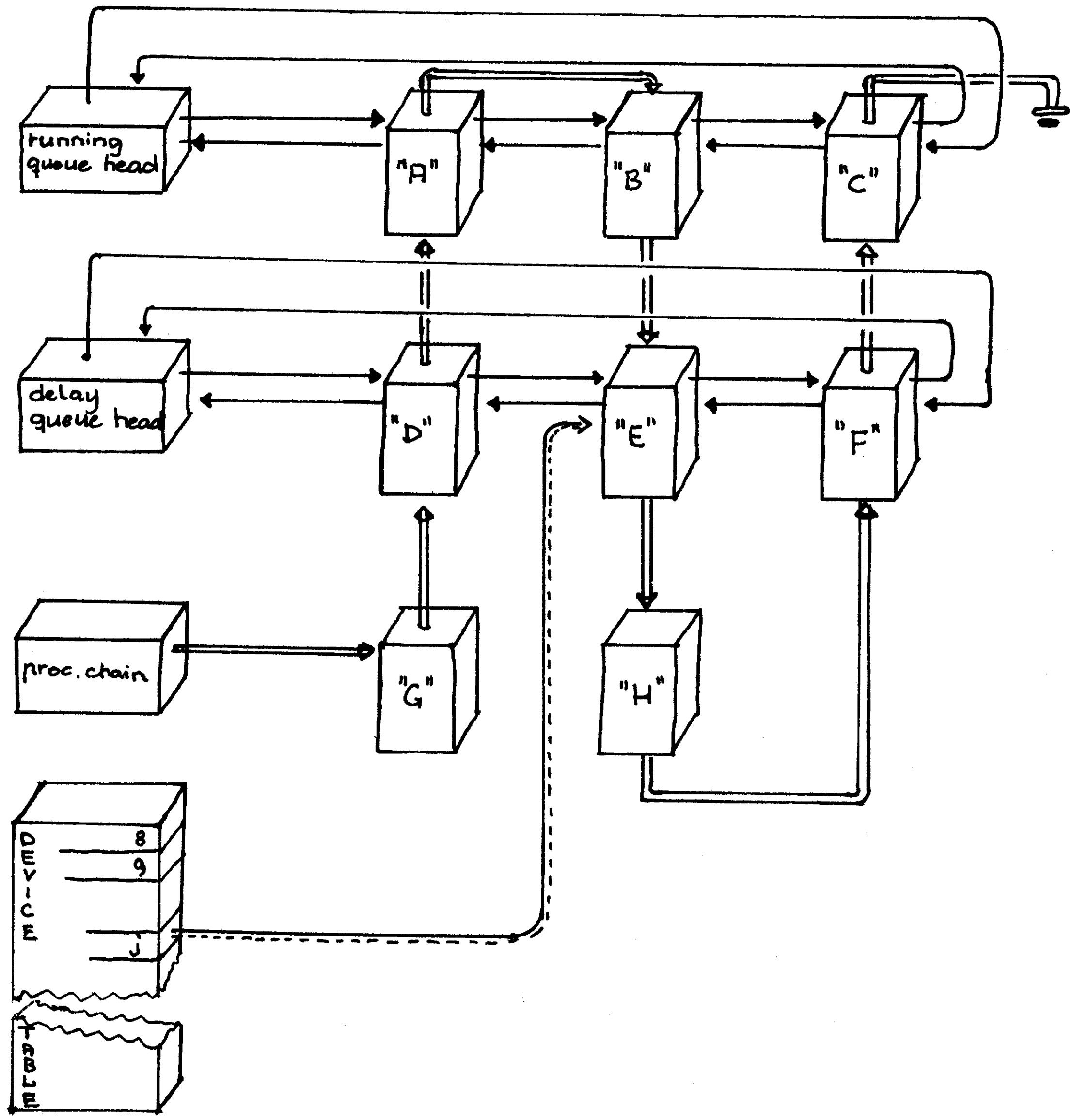


Fig. 2.2 Process descriptor organisation

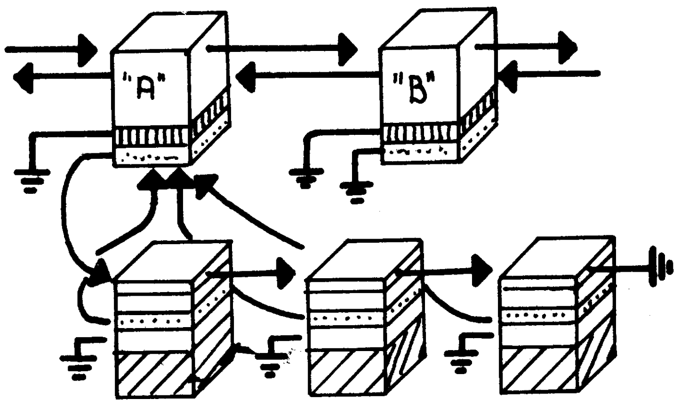
Process chain: G, D, A, B, E, H, F, C

Processes A, B and C are running, D, E and F are waiting for a delay to expire, E is also waiting for an interrupt from device j.G and H are stopped, without delay.

The processes G and H may be waiting for an event , and process E may be waiting for a general event.

Notice that the monitor imposes no restrictions on the processes which communicate with devices. Special device handlers do not exist within MUS-monitor.
The term driver process is used to describe a normal process, which is dedicated to operation of a device. They have been introduced in order to give a logically clean approach to I/O-handling.

Communication facilities for internal processes are designed with the concepts of a sender process, which sends a message to a receiver, which in turn returns an answer.
Messages and answers are fixed amounts of information, placed in special message-buffers and moved between processes by monitor functions. These buffers are part of the process and belong to the process, but should not be used directly by it. This method which takes common information away from the code should ensure, that programming errors or misunderstandings about the communication procedures to a reasonable degree, should have only local effect.



running queue links
event queue head
buffer chain head

owner chain
auxiliary links
sender link (permanent)
receiver link
information

fig. 2.3 a Initial no communication state

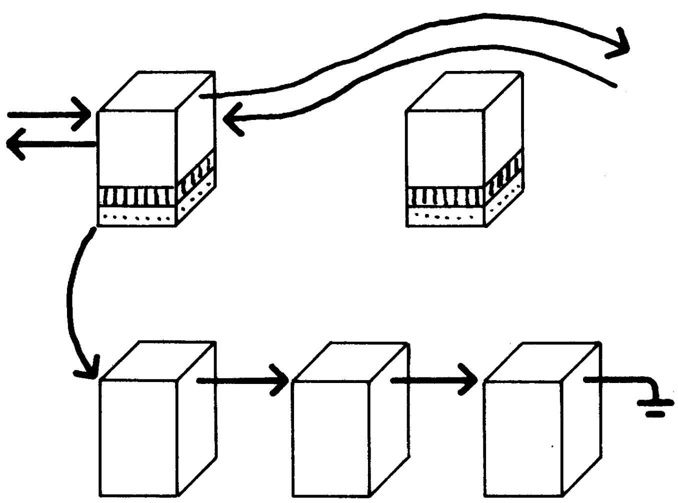


fig. 2.3 b B waits for a general event

Process B is linked out of running queue, as no event is pending, it is set in state waiting.

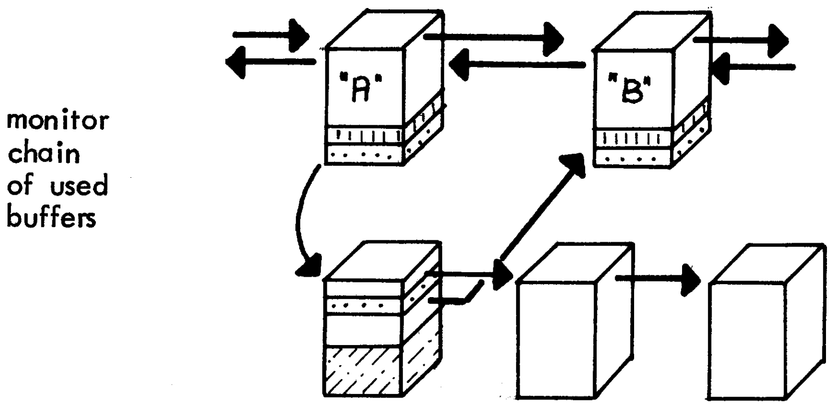


fig. 2.3 c A sends a message to B

First free buffer is loaded with information, and linked to eventqueue of B. Receiver address is put to B.

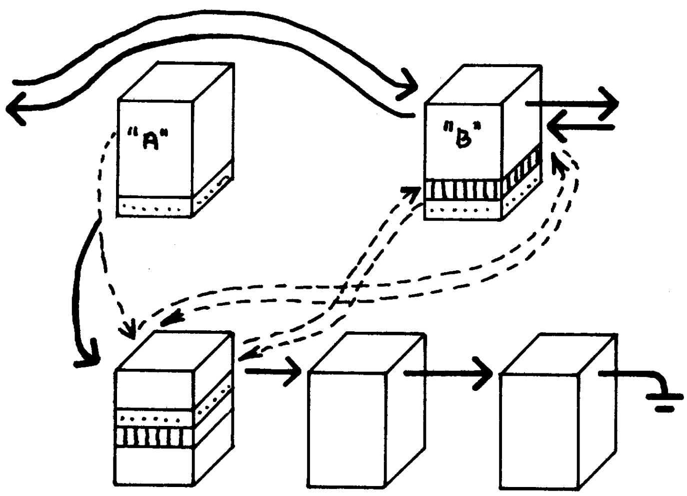


fig. 2.3 d: A waits for answer (specific event)

Process A is stopped, as no answer to the message has appeared.

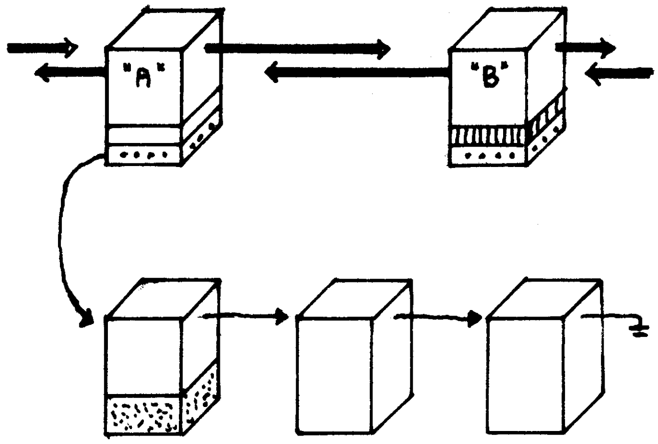


fig. 2.3 e: B sends the answer.
The buffer is removed from the eventqueue of B, and is momentarily linked to the eventqueue of A. As A is waiting for the answer, the buffer is removed from the eventqueue of A and set into the free buffer pool of A.

List of elementary monitor functions:

wait-interrupt (device, delay), waits for interrupt from device. When the delay period has expired the process is started unconditionally.

send-message (information, receiver-name), copies the information into a free message buffer and links it to the receiver eventqueue.

wait-event (information, bufferaddress). If bufferaddress is zero it waits for an event (message or answer) to arrive in the eventqueue of the process. Otherwise bufferaddress should point at a buffer in eventqueue, and the function waits for arrival of an event after this buffer.

send-answer (information, bufferaddress), puts information into the buffer addressed and returns it to sender.

wait-answer (information, bufferaddress), is a special version of wait-event, which waits for a specified buffer and when it arrives collects the information in it and returns it to free buffer pool.

wait (device, delay, bufferaddress) combines the function of waitinterrupt and waitevent.

Other features of the monitor.

Besides process chain, two further chains are kept by the monitor. Program chain which chain all program areas together and Free core chain which contains all unused core areas.
In this way all of core belongs to a chain, which can be process chain, free core chain, monitor used buffer chain or the separate free buffer chains of the processes.

Special attention has been paid in the implementation to the problem of reentrant programs. All data areas can be placed as part of the process descriptor, the address of which may be loaded by a single instruction anywhere in program.
This is a very convenient way to eliminate programmer kept data segment pointers.

DRIVER PROCESSES

A driver process is a normal process seen from the monitor.

The reasons for introduction of process dedicated to device control are:

- to let more than one process communicate with a device. Without a driver as interface, this would demand explicit arrangements among involved processes.

- to handle devices in a more uniform way. That is introduction of standard operations, standard status information. Blocking of all input/output, also for character oriented devices.
- to realise simple conversions of characters directly from input or output to the device..

The operations may be split into two classes,

1. control operations which does not imply any actual input or output, but which performs positioning, selects different facilities etc.

2. transput operations which calls for input or output to a core area.

The operations are communicated through messages to the driver process. Regardless of the operation the answer received when a message has been treated by the driver process contains a status word, which describes how the execution went.

Messages

The formats for messages are:

control:

mode	X 0
special 1	
special 2	
special 3	

transput:

mode	Q 1
bytecount	
first byte	
special	

Operation is the common term used for the first word. It is split into a 14 bit mode and a 2 bit basic command.

If b15 is zero, it is a control operation, otherwise a transput operation. A transput may be either input, Q = 0 or output, Q = 1.

Modebits of a control operation are used to specify control actions. The following actions exist at the moment.

Reservation: the driver is reserved for exclusive access by the sender process, or reservation is released.
Conversion: a conversion table address is set up in the driver process. The format of conversion tables is driver dependent. Note that if conversion is used the driver should be reserved, otherwise one cannot be sure that the proper conversion table is used. Another process may have specified its own.
Termination: is used to close output logically. E.g. a tapemark may be written on magnetic tape.
Position: specifies the execution of a positioning operation for devices which can be positioned.
Disconnection: means that device should be set offline if possible.
Erasure: is used to delete previous output on magnetic tape for example.

Not all mode-actions may be relevant for a specific driver process.

Modebits of a transput operation are used entirely in driver dependent fashions.

Special words of messages are used in connection with the modebits.

Bytecount and firstbyte of a transput message defines the core area, which should be input or output.

Answers

All answers from a driver have a standard format:

status
byte count
special 1
special 2

Status is an array of 16 bits, with standard interpretation:

bit:	interpretation, if set:
b0:	device disconnected
b1:	device off-line
b2:	device not-ready
b3:	device mode 1
b4:	device mode 2
b5:	device mode 3
b6:	illegal message or device reserved by other process
b7:	end-of-file
b8:	block-error
b9:	data-late
b10:	parity-error
b11:	end-of-medium
b12:	position error
b13:	0
b14:	timer error
b15:	0

The status bits b0, b1, b2, b6, b7, b8, b9, b10, b11, b14, are called clean bits. It means that if they occur, the driver shall return all following transput messages with status = 0 and count = 0.

Bytecount is the number of bytes actually input or output.

Special words may be used to give a document position.

I/O-HANDLING

The reentrant i/o procedures, which may be included in the MUS system, work on zones. A zone is a collection of information and associated storage areas necessary to perform operations on documents (data sets).

A zone contains 3 parts: Zone descriptor, which contains information about the document and the device, that holds it. Share descriptors which holds information about the current activities in the buffers which they describe. A buffer area which physically contains the descriptors and associated buffers.

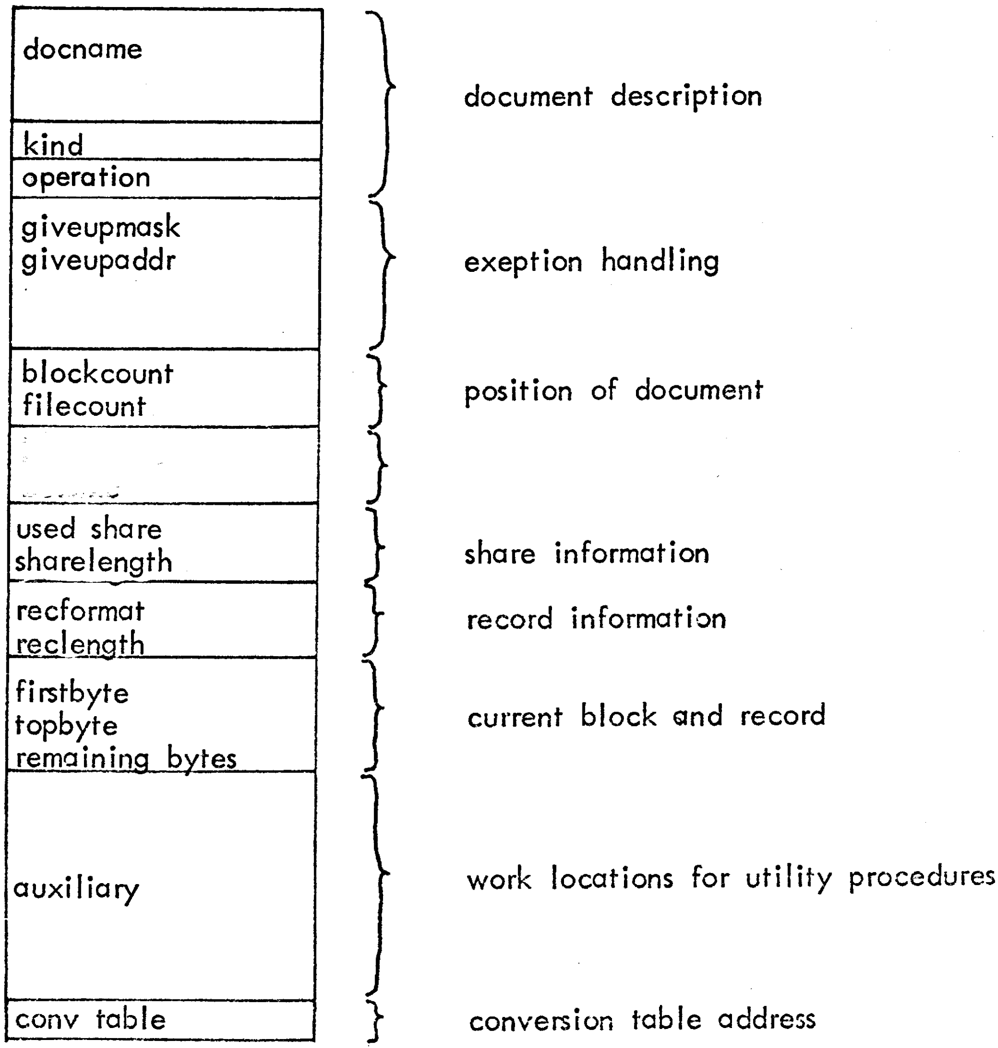


fig. 4.1 zone descriptor.

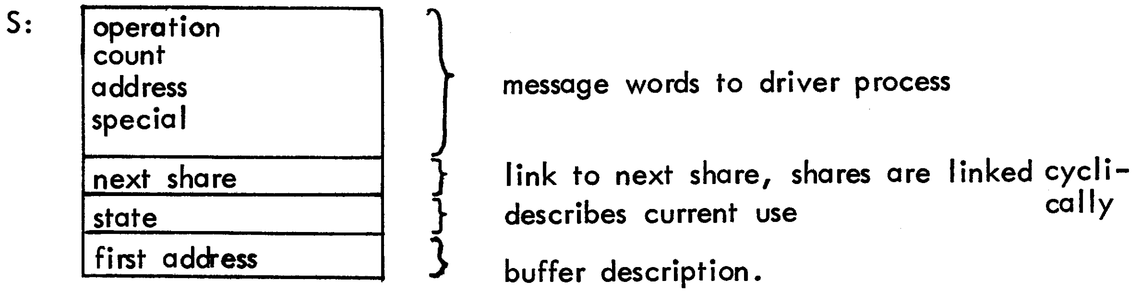


fig. 4.2 Share desriptor

Fu ll organisation:

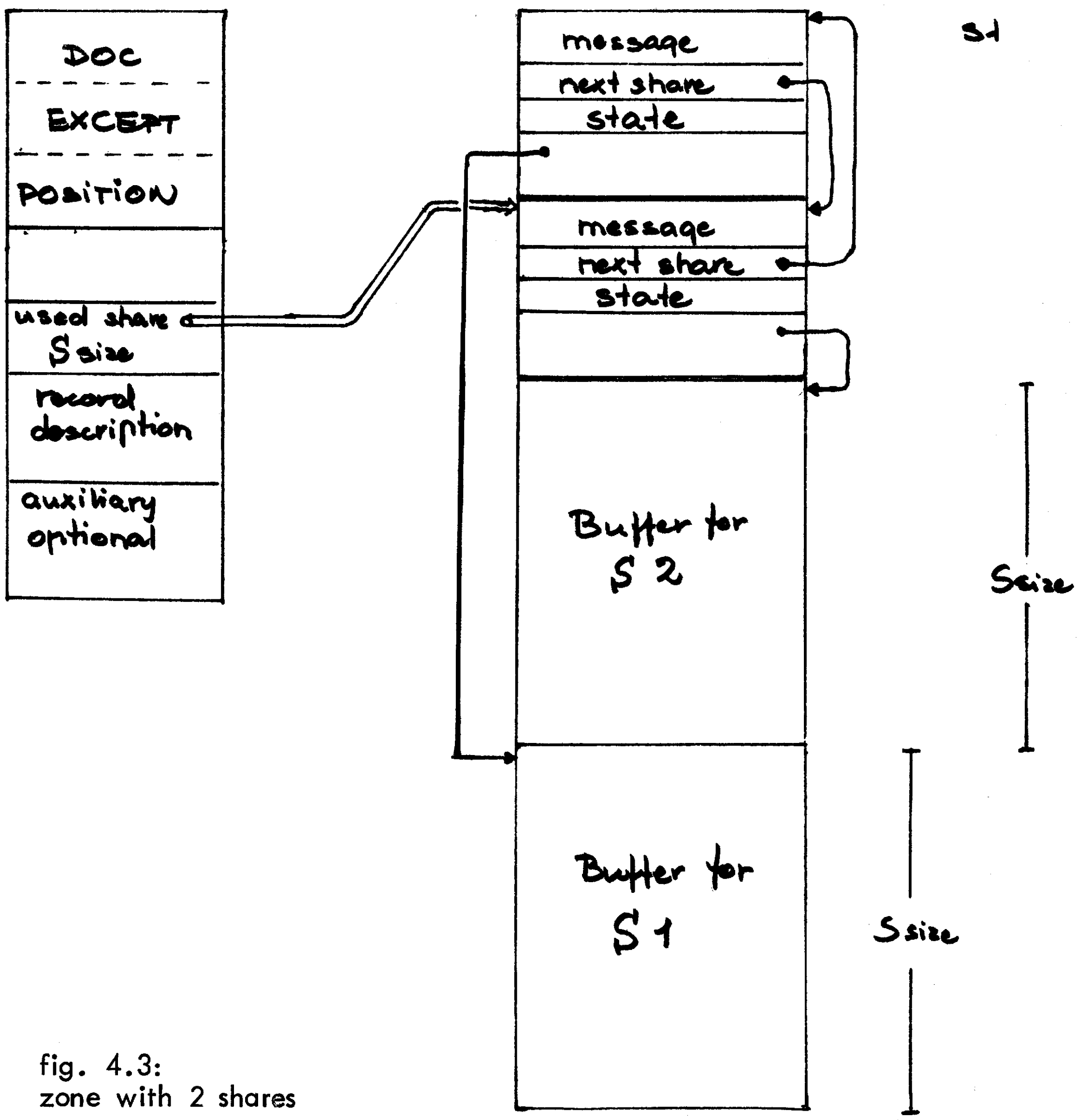


fig. 4.3:
zone with 2 shares

If a zone is to be set up in assembler code, the following parts should be initialized:

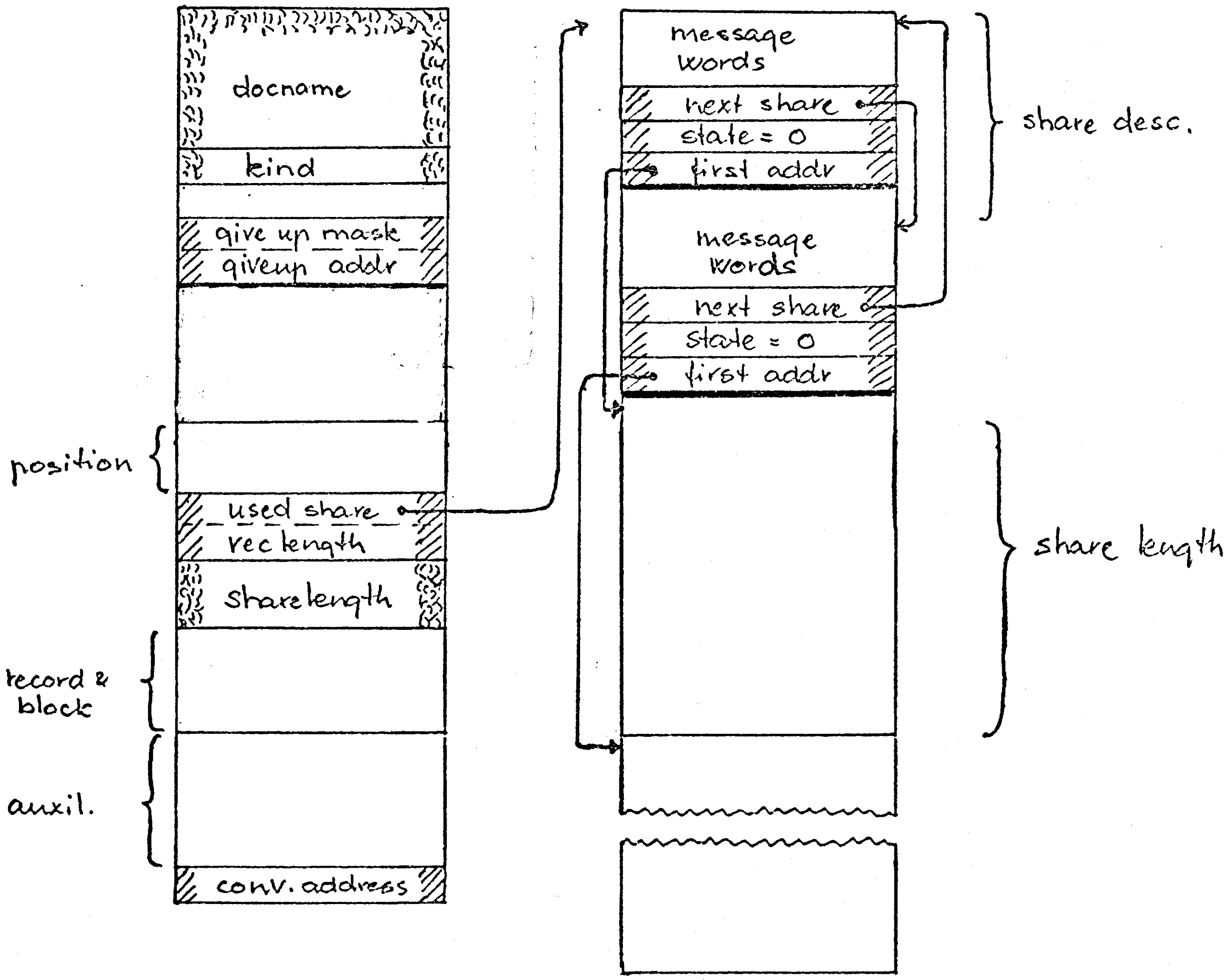


fig. 4.5 : zone and buffer after init.

I/O POSITIONING PROCEDURES

open (zone, operation)

sets operation of zone. This prepares later operation. Then it initialises the record and block information and sends a reservation message to the driver process. If conversion table address is different from zero a request for conversion is included.

setposition (zone, filecount, block count).

The values are placed in the zone, and a control message specifying position is sent to the driver process.

close (zone, release).

Outputs a last block if necessary. Wait for all pending transfers, which may have been initiated by the transfer procedures. If the second parameter is nonzero a release and disconnect message is sent to the driver process. If command part of operation = 3 (output) a termination message is sent independent of the second parameter.

I/O TRANSFER PROCEDURES

This subset of the I/O procedures falls into three classes. One is the basic block transfer procedures common for the remaining procedures. The second is the character oriented procedures, which transfer information in character form. The third is the record oriented procedures which transfer information in terms of records of various formats.

Block-oriented procedures

transfer (zone, length, operation);

A operation is started in used share.

waittransfer (zone).

If state of used share is 0 (free) the procedure is dummy. Otherwise it waits for a pending message (initiated by transfer) and adjusts the zone parameters: remaining of block and top address, which describes the block input or output. Then the transfer is checked using the status and givupmask.

Note: use of these primitive transfer procedures, should not be common practice. They should only be used if a special bufferadministration is wanted.

inblock (zone).

Starts input of one or more blocks to the available share buffers according to a circular buffer-strategy. Then it waits for a single operation to be finished, ready for use.

outblock (zone).

Makes the next share buffer available for output, after having started an output operation for the current one.

Character I/O procedures.

inchar (zone, char)

makes the next character from the zone available.

outchar (zone, char).

outputs the character on the specified zone.

outend (zone, byte)

works as a close with no release on character oriented devices, otherwise as outchar.

outtext (zone, textaddr)

outputs a text terminated by a Null-character by means of outchar. The Null-character is not output.

outoctal (zone, integervalue)

outputs a 16 bit binary integer in octal form, as 6 ASCII characters.

Record oriented procedures

getrec (zone, length, recaddress);

makes the next record as determined by recordformat of zone available at recordaddress and onwards. The length of the record must be given for U-formats and is always returned in the length parameter.

putrec (zone, length)

makes room for the record specified by length.

RECORD FORMATS

The items of data in a document are arranged in blocks separated by interblock gaps (IBG); a block is the unit of data transmitted to and from a document. Each block contains one record, part of a record or several records; a record is the unit of data transmitted to and from a process.

If a block contains two or more records, the records are said to be blocked. Blocking conserves storage space on the physical medium containing the document because it reduces the number of interblock gaps, and it may increase efficiency by reducing the number of input/output operations required to process a data set. Records are blocked and deblocked automatically by procedures getrec and putrec.

The records in a data set must be in one of three formats: fixed-length, variable-length, or undefined-length. They can either be blocked or unblocked. The following paragraphs describe the three record formats.

FIXED-LENGTH RECORDS

In a document with fixed-length (F-format and FB-format) records, (see Figure 4.6) all records have the same length. If the records are blocked, each block contains an equal number of fixed-length records (although the last block may be truncated if there are insufficient records to fill it). If the records are unblocked, each record constitutes a block. If the blocklength is not an integral multiple of the recordlength, some space is left unused in the block.

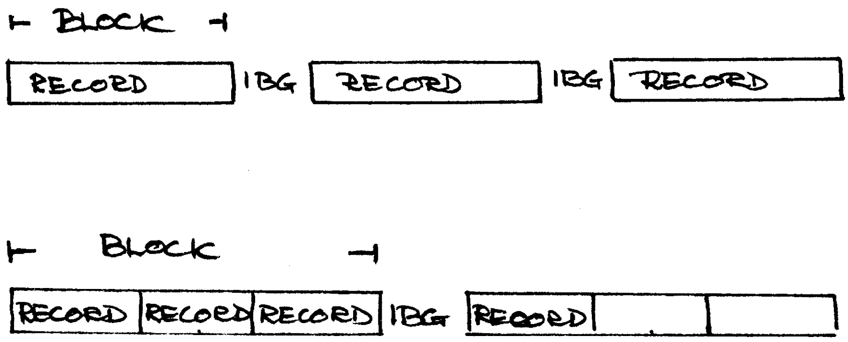


fig. 4.6 Fixed-Length Records

VARIABLE - LENGTH RECORDS

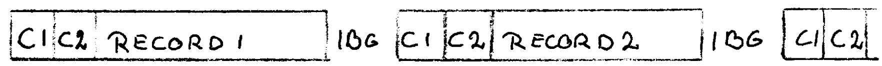
This format permits both variable-length records and variable-length blocks. The first four bytes of each record and of each block contain control information for use by the procedures (including the length in bytes of the record or block). Variable-length records can have one of two formats:

V, VB (figure 4.7)

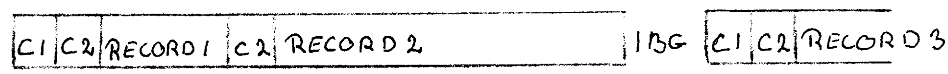
V-format signifies unblocked variable-length records. Each record is treated as a block containing only one record, the first four bytes of the block contain block control information, and the next four contain record control. VB-format signifies blocked variable-length records. Each block contains as many complete records as it can accomodate. The first four bytes of the block contain block information, and the first four bytes of each record contain record control information.

Fig. 4.7 a:

V FORMAT



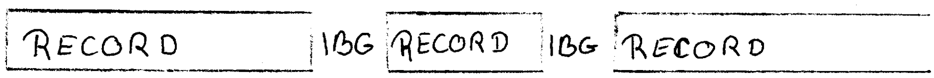
VB FORMAT



UNDEFINED - LENGTH RECORDS

In this format a record is either an entire block, in unblocked format, or a number of bytes of the block in bloked format (see figure 4.8). The user must detemine the number of bytes wanted for a record.

Unblocked records (U-format):



Blocked records (UB-format):

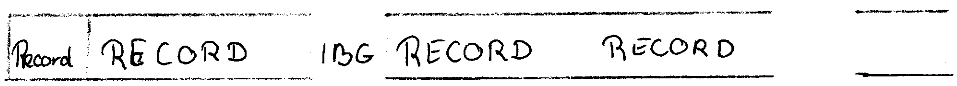


Fig. 4.8: Undefined-length records.

OPERATOR COMMUNICATION

Within a computing system, which contains a single process communicating with the human operator, there is no real problem in this communication. All the process needs to know is the device for output and the device for input of concern to the operator (actually it may be the symbolic names of the associated drivers).
When more than one process wants to communicate with the single operator an identification problem arises. How is the operator to distinguish messages from different processes, and how is he sure that a reply reaches the correct process?
The answer to these questions within MUS is introduction of operator processes, which on one side communicates with the human operator through the operator devices and on the other side acts as operator for the processes.

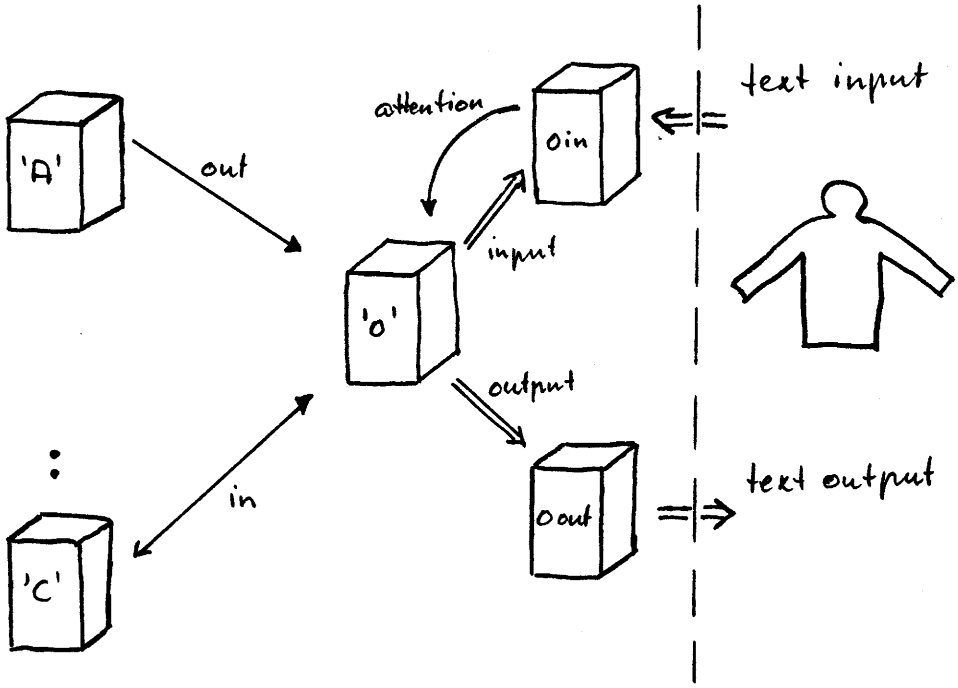


Fig. 5.1 : 'o'functions

OPERATING SYSTEM

The human operator has two distinct roles to play within a processing system. One is to serve the system when it calls for something to be done (eg. mount a tape, change paper in printer, supply parameters to a program); the other is to act as master for the system (eg. load programs, create processes, start processes).

Within MUS communication with the serving operator is a matter which the single process must take care of, but the master operator has to have some means to carry out his commands. This is precisely the reason for introduction of an operating system process "S", which can effectuate master operator commands.

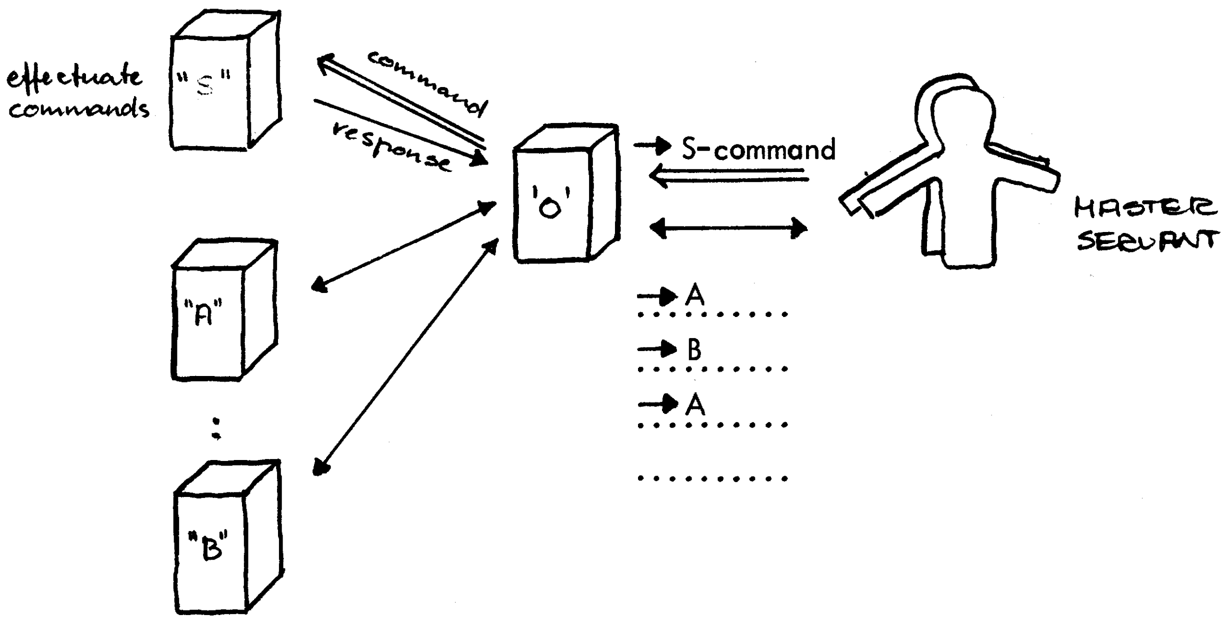


fig. 6.1 : "sys" process and operator

Commands for "sys" are single lines of text, which should conform to the following syntax:

CALL [MODIF MODIF .....]

CALL determines the basic function, and MODIFs qualifies the execution.

RCSL : 44 - RT - 759
Date : Rev. June 1975
Authors: Torkild Glaven
AP Ravn

MUS

PROGRAMMERS GUIDE

(part two of two)

Keywords: Multiprogramming, monitor, devicehandling, i/o-utility, record i/o, operator communication, operating system.

Abstract: The manual is mainly intended for readers who are going to use the system. The user is assumed to be familiar with the general principles of the system as well as with the assembler language.

This manual obsoletes RCSL 44 - RT 508 RC 7000 System Software Nucleus.

CONTENTS PAGE

INTRODUCTION

Without the monitor we have the cpu operating in parallel with the devices. Only one program can run in the cpu so we have one process running in parallel with the devices. This process is able to communicate with the devices by means of io instructions and the interruption system.

Multiprocessing

The primary purpose of the monitor is to implement multiprocessing., i.e. simulate multiple processes running in parallel by sharing the cpu and the devices.

In order to implement the advanced tool of cpu time-sharing the monitor uses the two primitive tools:

real time clock device and

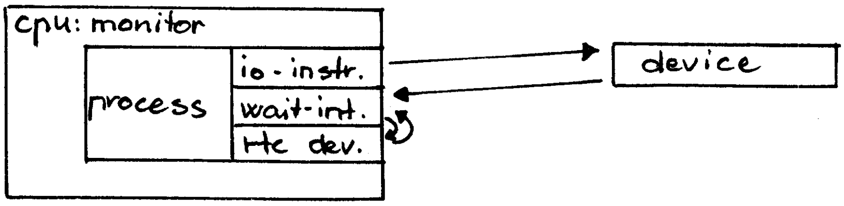
interruption system.

Having occupied these facilities the monitor must supply the process with corresponding facilities.

The monitor thereforesimulates that each process is supplied with a real time clock device. This device gives an interrupt after a real time delay specified by the process.

The monitor also supplies the process with an interruption facility, the monitor function: wait interrupt. This enables each process to wait for interrupt from any device except the cpu, but including the simulated real time clock device. Interrupt from the cpu, power failure interrupt, is not available for the processes. When it occurs the processes are breaked and informed about the cause.

Now we have a number of processes running in parallel with the devices. Each process is able to communicate with its own clock device and all other devices. Processes are unknown to each other.



Monitor Functions

One monitor function has already been introduced: wait interrupt. Monitor functions perform indivisible operations on tables, queues, chains, etc. The functions are called by the processes and executed by the monitor in disabled mode. Seen from the processes they are extended instructions. The total list of monitor functions is:

Interruption:

- wait interrupt

Process Communication:

- send message
- wait answer
- wait event
- send answer

General Communication:

- wait

Operating System Facilities:

- search item
- clean process
- break process
- stop process
- start process

Process Communication

The four monitor functions for process communication enable the processes to exchange information by means of message buffers (shortly: buffers). Each process has a pool of unused buffers. At present a buffer contains a head of 6 words and an information part of 4 words. A communication takes place in the following way: The sending process sends a message to the receiving process by means of send message. The receiver gets information about the message by means of wait event. The receiver returns the buffer as an answer by means of send answer. The original sender may get information about the answer by means of wait event, before the buffer is released by means of wait answer. If the sender wants to wait for answer to a specific message, it suffices to use wait answer.

General Communication

The function, wait, works as a combined waitinterrupt and waitanswer. In this way it is possible to wait for an interrupt or a timeout or an event.

Operating System Facilities

The monitor function, search item, searches for a named item in a specified chain.

The monitor function, clean process, is performed on all processes after a power failure. The function cleans the communication situation and breaks the processes.

The monitor function, break process, is performed at monitor function call error. The process is started at its break address with an error number in a register.

The monitor function, stop process, sets a process in state stopped. If it is waiting, the program counter is decreased so the monitor function is performed again after start. The process is linked out of any queue of which it is a member.

The monitor function, start process, sets a process in state running and links it to running queue. This takes place if the state of process is stopped; otherwise the function is dummy.

FORMATS

Process Descriptor

A process descriptor is an item. Each process has a process descriptor containing important process parameters such as name, state, and saved registers.

next.proc: next process in a queue of processes.

prev.proc: previous process in a queue of processes.

This queue element links the process to the running queue or to the delay queue, or it points at itself.

chain.proc: next process in the process chain.

All process descriptors are in this chain.

size.proc: process descriptor size.

Process descriptors are of variable lengths.

name.proc: process name (three words).

The process is identified by this text of one to five characters, unused character positions equal zero.

event.proc: first event in event queue.

+1: last event in event queue.

This queue element is the event queue head. The queue contains messages and answers to the process.

buffe.proc: first message buffer.

Message buffer chain head. The chain contains the message buffers belonging to the process.

prog.proc. program address.

Address of the program executed by the process. A program may be used by one or more processes.

state.proc: process state.

-8-63 waiting for interrupt, event or software timer
-2 waiting for event or software timer
-1 waiting for event
0 running (i.e. linked to running queue) or waiting for software timer
8-63 waiting for interrupt from device no = state
buf > 63 process waiting for answer in that buffer
1b0 process stopped

timer.proc: timer count.

The number of timer periods the process still will wait in the delay queue.

prior.proc: priority.

Priorities are unsigned values (zero must not be used). Current process (executing instructions) is chosen cyclically among the processeswith highest priority.

bread.proc: break address.

This address is entered after an operator break , a power failure, or a program error. It must always be defined.

ac0.proc: saved ac0 .

ac1.proc: saved ac1.

ac2.proc: saved ac2.

ac3.proc: saved ac3 .

psw.proc: psw (process status word) = pc * 2 + carry.

When the process is not active the registers are saved here.

save.proc: work location for basic reentrant procedures.

o.proc: process optional words.

The process descriptor may contain any number of optional words.

E.g:

The optional words are used by the driver utility procedures, as:

buf.proc: saved message buffer address.

addre.proc: current value of address.

count.proc: current value of count.

reser.proc: process descriptor address of reserving process.

Zero indicates no reserver.

convt.proc: conversion table address.Zero indicates no conversion.

clint. proc: interrupt handling entry address. This address is entered in disabled mode, when an interrupt arrives from a device, which the process wants to supervise.

This means that a driver process should contain at least 6 optional words, if it wants to utilize the procedures.

Message Buffer

A message buffer is an item. Its head of 6 words contains the item head and references to the sending process and the receiving process. The remaining part contains the transmitted information.

next.buf: next buffer in a queue of buffers.

prev.buf: previous buffer in a queue of buffers.

This queue element links the message buffer to the event queue of a process, or it points at itself.

chain.buf: next buffer in a chain of buffers.

All message buffers of a process are chained together.

size.buf: size of the buffer.

At present the size equals ten.

sende.buf: sender process descriptor address.

This value is permanent.

recei.buf: receiver parameter.

buffer state:	receiver parameter value:
free	0
message (not yet answered)	+ receiver process descriptor address
answer	- receiver process descriptor address

The next words have optional contents depending on the use, for example:

mess0.buf:	operation	status word
mess1.buf:	byte count	byte count
mess2.buf:	first word address	file number
mess3.buf:	special information	block number

The format of an input/output message to a driver is defined in the driver description. A few standards are used:

The first word contains the operation. Which is split into a 14 bit mode and a command. Operation(15:15) defines a control message (=0) or a transput message (=1). For transput messages operation (14:14) defines input (=0) or output (=1). The second word normally contains byte count, the third word normally contains first word address, and the fourth word has a special content depending on the operation and driver.

Answer from a driver normally contains the status word and the number of bytes transferred in the first two words. Further specification is found in the driver description.

Program

A program is an item of the program chain. The program head contains information about the size and name of program and a descriptor word.

pspec.prog: program descriptor word.
pstar.prog: start address for program.
chain.prog: link to next program in chain.
size.prog: size of program.
name.prog: program name (three words),
The program is identified by this text of 1 to 5 characters.

The program descriptor word is an array of bits, which describe the use of the program.

b0: own bit, if set, the program contains its own process descriptor after the program. This process descriptor is used, if the program should be started as a process.
Thus the process descriptor address is prog+size.prog.
b1: reentrant bit, if set the program is reentrant.
b2: page zero bit, if set the program uses page zero locations.
b8-b15: process count, the number of existing process descriptors, which use this as program.

Free Core area

A free core area is an item of the free core chain. At present the items of the chain cannot be handled by any standard procedures. In later versions of the system they may be used for dynamic storage allocation.

Catalog

A catalog entry is an item of the catalog entry chain. At present only the entry head exists.

Page Zero Locations

The monitor leaves about half of page zero, 128 (decimal) locations, for use by user programs translated by compilers.
It is strongly emphasized, that the system is not prepared to take care of programs using page zero locations, this is at own risk in the multiprogramming system.

Monitor Process Descriptor

The monitor is organized as a process which process descriptor contains all tables and the code for the monitor. However the locations 0-31 are outside this process descriptor. They are used in the following way:

0-1 :	interruption system
2-13:	monitor function entries
14-15:	two page zero locations to be used in disabled mode by processes.
16-17:	two autoincrementing locations to be used in disabled mode by processes.
18-29:	monitor function references.
30-31:	two autodecrementing locations to be used in disabled mode by processes.

The monitor process has the lowest possible priority (zero) which must not be used for other processes. So the monitor is active as a process only when no other process wants to execute instructions. The monitor process executes a dummy program: jmp .+0 in enabled mode.

Only the first part of the monitor process descriptor, corresponding to a normal process descriptor, is described here. Some of its parameters act as normal process parameters in order to let the monitor run as a dummy process when no other processes wants execution time. The remaining locations are used for important monitor constants and variables.

cur: first process in running queue.
+1: last process in running queue.

Head of running queue and process chain. A process may always find its process descriptor address (current process descriptor address) in cur.

opera:	reference to name of operator process
size:	monitor process descriptor size.
table:	device table.
	Contains a word for each device number holding process descriptor address for interrupt requesting process.
topta:	top of device table.
runni:	running queue.
	Reference to head of running queue.
proce:	process chain.
	Reference to monitor process chain.
monit:	monitor process description.
	Reference to monitor process descriptor address.
dfirs:	first process in delay queue.
+1:	last process in delay queue.
	Head of delay queue and message buffer chain.
pfirs:	first in program chain.
exit:	monitor exit address.
efirs:	first in entry chain.
ffirs:	first in free core chain.
delay:	delay queue.
	Reference to head of delay queue.

PAGE ZERO VARIABLES

The page zero variables are part of the monitor process descriptor

cores: core size.

Contains the number of words in core.

frequ: frequency of rtc.

Defines the real time clock frequency:

0:	50 hz
1:	10 hz
2:	100 hz
3:	1000 hz

progr: program chain.

Reference to head of program chain.

entry: entry chain.

Reference to head of entry chain.

free: free core chain.

Reference to head of free core chain.

mask: interrupt mask

PAGE ZERO CONSTANTS

The page zero constants are part of the monitor process descriptor. These currently used constants are placed in page zero in order to decrease program core requirements.

Bit patterns

The bit patterns, 1b0, 1b1, ..., 1b15, are placed in consecutive locations labelled by a point and the value, for example:

```
.1b12:    1b12
```

The first location has a further label, bit, so if ac2 equals 7, the instruction,

```
lda   0   bit,2
```

loads the bit pattern, 1b7, into ac0.

Decimal constants

Now follows a list of decimal constants available for the programs, but not necessarily placed in the here shown order:

```
.0:     0
.1:     1
.2:     2
.3:     3
.4:     4
.5:     5
.6:     6
.7:     7
.8:     8
.9:     9
.10:    10
.12:    12
.13:    13
.15:    15
.16:    16
.24:    24
.25:    25
.32:    32
.40:    40
.48:    48
.56:    56
.60:    60
```

.63:	63	
.64:	64	
.120:	120	
.127:	127	
.128:	128	
.255:	255	
.256:	256	
.512:	512	
.1024:	1024	
.2048:	2048	
.4096:	4096	
.8192:	8192	
.16384:	16384	
.32768:	32768	
.m3:	-3	
.m4:	-4	
.m16:	-16	
.m256:	-256	
.name:	name	(relative address of name in item)
.mess:	mess0	(relative address of mess0 in buf)
.even:	event	(relative address of event.proc)
.z:	z	(standard zone size)
.ssiz:	ssize	(size of a share descriptor)
.NL:	10	
.CR:	13	
.LF:	10	
.FF:	12	

Status bits

sdisc:	1b0	(disconnected)
soffl:	1b1	(offline)
sbusy:	1b2	(busy)
sdev1:	1b3	(device mode 1)
sdev2:	1b4	(device mode 2)
sdev3:	1b5	(device mode 3)
sille:	1b6	(illegal)
seof:	1b7	(end of file)
sbloc:	1b8	(block error)
sdata:	1b9	(data late)
spari:	1b10	(parity error)
sem:	1b11	(end medium)
s12:	1b12	(position error)
snotp:	1b13	(rejected by wait transfer)
stime:	1b14	(timer)
s15:	1b15	(hard error in wait transfer)

Control bits

ceras:	1b8	(erasure)
cdisc:	1b9	(disconnect)
cposi:	1b10	(positioning)
cterm:	1b11	(termination)
cconv:	1b12	(conversion)
crese:	1b13	(reservation)

MONITOR FUNCTIONS

The functions are called from assembler code by writing their names. Link is automatically defined. The functions are executed in disabled mode by the monitor.

In case of parameter error in call, current process is breaked with the error number (always negative) in ac0. If the function is not implemented, the calling process is breaked with error number = -1.

The functions are described in the following. The return value of ac3 (cur) is the process descriptor address of the calling process (current process).

Function Wait Interrupt (device, delay)

	call:	return:	link	
ac0		unchanged	+0:	timeout
ac1	device	device	+1:	interrupt
ac2	delay	cur		
ac3	link	cur		

The corresponding entry in devicetable is checked for an interrupt.
If interrupt is pending return is made immediately to (link +1).

Delay is inserted as timer count in the process descriptor and the process is linked to the delay queue. If delay is zero a maximum waiting period is specified.
The process is stopped with status = waiting for interrupt or software timer.

Return depends on the event: If the time specified by delay runs out before the interrupt arrives, return is performed to time out (link+0), otherwise to interrupt (link+1).

Note: Wait Interrupt may be used as a pure timer, when device = 0.

Note: Before any call of wait interrupt with device ≠ 0, the device table entry must be initialized to proc * 2.
This may be done by procedure setinterrrupt.

Function Send Message (address, name address, buf)

	call:	return:
ac0		unchanged
ac1	address	address
ac2	name address	buf
ac3	link	cur

Selects a free message buffer belonging to the calling process and copies the message at address and on into this message buffer (4 words). The message buffer is delivered into the queue of a receiving process with name placed at name address and on. The receiving process is activated if it is waiting for an event. The calling process continues execution after being informed about the address of the message buffer.

The format and interpretation of a message depends on the receiving process.

Errors:

-2: There exists no process with the given name.

-3: No free message buffer is available at the moment.

Function Wait Answer (first, second, buf)

	call:	return:
ac0		first
ac1		second
ac2	buf	buf
ac3	link	cur

Delays the calling process until an answer arrives in the message buffer given as parameter. The process is supplied with the first two words of the answer. The message buffer is released.

The format of the answer depends on the process that has answered the message.

Errors:

-2: The message buffer address does not point at a message buffer belonging to the calling process.

Function Wait Event (first, second, buf)

	call:	return:	link	
ac0		first	+0:	answer
ac1		second	+1:	message
ac2	buf	buf		
ac3	link	cur		

Delays the calling process until an event (a message or an answer) arrives in its queue after the message buffer given as parameter. If this parameter is zero, the queue is examined from its beginning. The calling process is supplied with the address of the event and with the first two words of the event.

Return depends on the event: If the event is an answer return is performed to answer (link+0), otherwise to message (link+1).

Errors:

-2: The message buffer address is neither zero nor pointing at a message buffer in the queue of the calling process.

Function Send Answer (first, second, buf)

	call:	return:
ac0	first	first
ac1	second	second
ac2	buf	buf
ac3	link	cur

The calling process delivers a first and a second word, which are copied into the first two words of the message buffer given as parameter. The message buffer is delivered as an answer in the queue of the sender.

Errors:

-2: The message buffer address does not point at a message buffer in the queue of the calling process.

Function Wait (delay, device, buf, first, second)

	call:	return:	link	
ac0	delay	(first)(unchanged)	+0:	timeout
ac1	device	(second)(device)	+1:	interrupt
ac2	buf	(buf)(cur)	+2:	answer
ac3	link	cur	+3:	message

Performs the combined functions of wait interrupt and wait-event.
Delay is inserted as timer count in the process descriptor, and the process is linked to delay queue. If device is non-zero, the devicetable is ohecked for an interrupt.

Then it waits for an event after the buffer given as parameter, if buf is zero the event queue is examined from the beginning.

If an event arrives first, return is made with the first two words of the message or answer and address of the buffer.
Otherwise the contents of the registers are as for waitinterrupt.
Errors:

-2: The message buffer address is neither zero nor pointing at a message buffer in the queue of the calling process.

Function Search Item (chain, name address, item)

	call:	return:
ac0		unchanged
ac1	chain	chain
ac2	name address	item
ac3	link	cur

If the chain contains an item with the name placed at name address and on, the address of this item is delivered, otherwise a zero is delivered.

Function Clean Process (proc)

	call:	return:
ac0		unchanged
ac1		unchanged
ac2	proc	proc
ac3	link	cur

Messages to the process are answered with status = not processed.
Answers to the process are released.
Messages from the process are released and the receivers are breaked, with error number = 1.
Finally the process is breaked with error number = 0.

Function Break Process (proc, error number)

	call:	return:
ac0	error number	error number
ac1		unchanged
ac2	proc	proc
ac3	link	cur

Error number should be greater than zero. The process is started at its break address with the following accumulator contents:

ac0:	error number
ac1:	unchanged
ac2:	proc
ac3:	psw //2 (its old program counter)

The following errornumbers are used by system procedures.

0:	clean process.
1:	clean process, message receiver.
2:	operator breaked process.
3:	end of program, MUSIL
4:	putrec, record too large, getrec illegal length of record.
5:	wait transfer, hard error.

Function Stop process (proc)

	call:	return:
ac0		unchanged
ac1		unchanged
ac2	proc	proc
ac3	link	cur

The process is set in state stopped and removed from delay- or running queue. If it was waiting for event or answer, psw is decreased by 2. This ensures, that the monitor function is called again if start process is performed.

Function Start Process (proc)

	call:	return:
ac0		unchanged
ac1		unchanged
ac2	proc	proc
ac3	link	cur

State of proc is examined. If it is stopped, the process is set running, otherwise the function is dummy.

DRIVER PROCESSES

A driver process is dedicated to communication with a device. Under special circumstances it might take care of several devices. E.g. teletype input and teletype output.
Other processes must then request the driver process to perform input/output operations. Driver processes are thus the only processes which actually execute i/o-instructions and call waitinterrupt.

Communication with other processes takes place via messages and answers. The messages should conform to the below mentioned standards, and the answers should also be of a standard form.
Note, that it is regarded as a rule, that all messages sent to a driver process should be answered in finite time.
Furthermore it is standard, that a driver process returns all waiting messages if a device operation goes wrong. This rule is a great help for the standard i/o-routine, when they use multibuffered input/output.

To code a driver program one should also be familiar with the standard recovery actions of the i/o procedures and with the document kind specification.

DEVICE HANDLING

Before any i/o instructions are executed, the driver process should clear the devices and insert its process descriptor address * 2 in the corresponding device table entries.
This may be done by procedure setinterrupt.

The driver process descriptor shall contain 6 optional words (see System Utility Procedures).
The last clint.proc shall give an address of an interrupt clear action.

clint.proc must obey the following conventions:

	called with:	return with:
AC0:		destroyed
AC1:	device	unchanged
AC2:	proc	unchanged
AC3:	link	destroyed

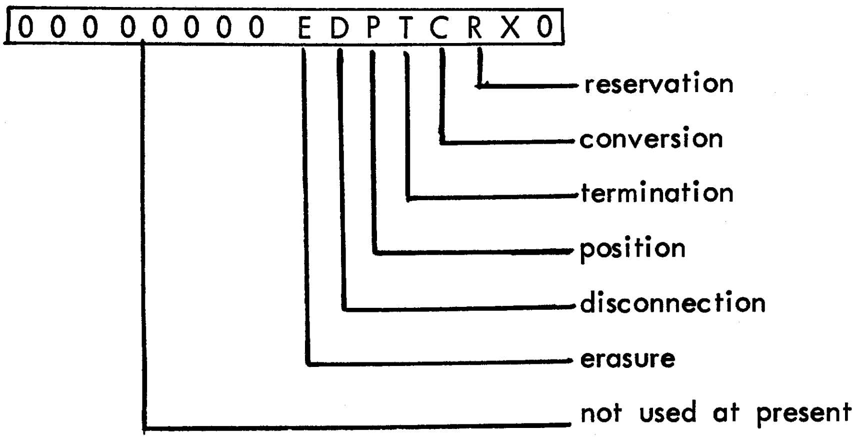
clint.proc is called in disabled mode and must not change this state.
It must return with the interrupting device cleared. The amount of data processing in clint.proc must be as little as possible since it affects system overhead, and clint.proc must never call other system procedures.
If only a nioc device is to be executed, the standard action clear may be used.
i.e. clint.proc: clear.

CONTROLMESSAGE

A controlmessage is used for a non-transfer i/o-operation. The format is:

mess0.buf:	operation
mess1.buf:	special1
mess2.buf:	special2
mess3.buf:	special3

Operation consists of mode (14 bits) and command.
The command specifies control (x0, bit 14 irrelevant).
Mode is an array of bits, which specify actions to be executed. An action is performed if the corresponding bit is one. Interpretation proceeds from bit 13 to bit 0. Not all actions are relevant for specific driver processes.



If a bit is set, the action is:
Reservation: If special–1 ≠ 0 the sender of the message gains exclusive access to the driver process. It is set as reserver in the process descriptor of the driver. Reservation means that messages from all other processes are returned with an illegal status, without being processed.
If special–1 = 0 a reservation is cancelled, that is the driver process will accept any messages again.

Conversion: Only relevant for character oriented devices. Special2 is used as address of a conversion table, which is placed in the process descriptor of the driver. A table address of zero specifies no conversion. Format and interpretation of the table is dependent on the driver. Note that if conversion is used, reservation ought to be done.

Termination: Only relevant for output devices.
The document which has previously been output is terminated logically. E.g. for a magnetic tape unit two file marks are written, and the tape is positioned between the two.

Position: The document is positioned according to the information in special2 (file count) and special3 (block count).

Disconnection: The device is set local (off.line).

Erasure: Only relevant for output devices, which are able to cancel pervious output. Special 1 may be used to specify how much that should be erased.

If all bits are zero only a sense command is executed.

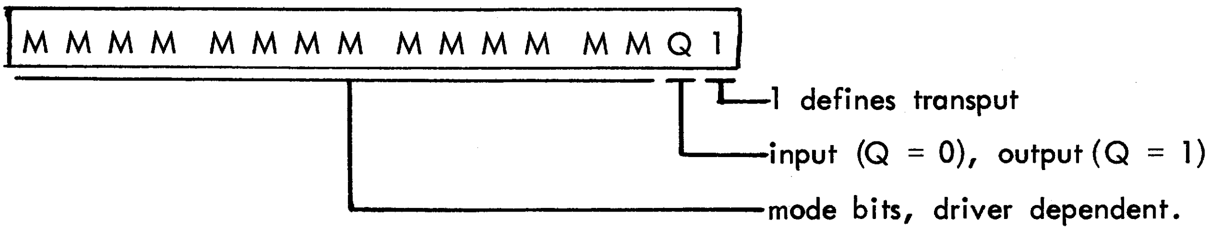
TRANSPUT MESSAGE

A transput message specifies an operation, which involves transfer to or from a core area.

The format of a message is:

mess0.buf:	operation
mess1.buf:	bytecount
mess2.buf:	first byte address
mess3.buf:	special

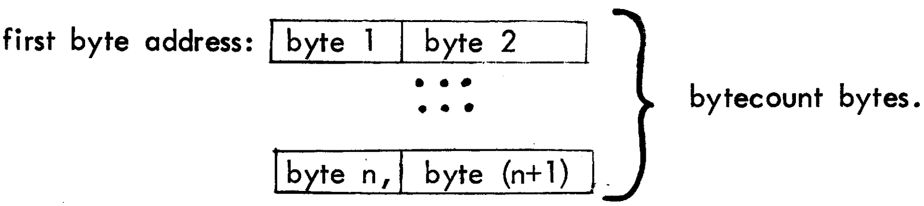
Operation consists of command and mode. Mode transmits information about the mode of transfer. E.g. odd parity, 7 track magnetic tape, decimal coded cards.



Byte count specifies the number of bytes to be transferred to or from core.

First byte address is the byte address of the first byte to be transferred.

The core area used for transfer is thus:



ANSWERS

The messages are independent of the command part of operation. The answer has the format:

mess0.buf:	status	(latest sensed status for control or transput message with count≠0 and status ≠b6 or bit 14)
mess1.buf:	bytecount	
mess2.buf:	speciala1	
mess3.buf:	speciala2	

Status is an array of bits, which convey information about device errors or call errors. The different bits have been given specific meanings in order to standardise error recovery in the input/output procedures.

b0:	disconnected,	*	the device is not present, power off for example.
b1:	off-line,	*	the device was off-line when an operation was attempted.
b2:	device busy,	*	the device was temporarily not able to execute the operation.
b3:	device mode 1		device dependent
b4:	device mode 2		device dependent
b5:	device mode 3		device dependent
b6:	illegal	*	the operation was rejected either because the driver was reserved by another process or because it was nonsense.
b7:	eof	*	logical end of document is detected (file mark, end of transmission sequence).
b8:	block error	*	the core area specified is too small to hold the block input.
b9:	data late	*	the high speed data channel responded too late.
b10:	parity error	*	one or more invalid characters were input in this operation.
b11:	end medium	*	physical end of medium. E.g. end-of-tape, paper tape reader empty, paper out for lineprinter.
b12:	0		not to be used
b13:	0		not to be used.

b14:	timer	* the device did not respond within a maximal time.
b15:	0	not to be used.

If a statusbit is marked * all immediately following transput messages should be returned with status zero. These bits are called clean bits.

Bytecount of answer specifies the number of bytes actually transferred.

Speciala1 is used for position information, (file count).

Speciala2 is used for position information, (block count).

SYSTEM UTILITY PROCEDURES

As an aid for the driver processes a number of actions, which frequently have to be executed, are collected as reentrant routines.

If they are used, the process descriptor should contain the following optional words:

buf.proc:	buffer address of current message
addre.proc:	value of mess2, first byte address
count.proc:	value of mess1, bytecount
reser.proc:	word containing reserver process.
convt.proc:	conversion table address.
clint.proc:	interrupt clear action address.

Procedure Next Operation (mode, count, buf)

	call:	return:	link	
ac0		mode (=operation(0:13))	+0:	control
ac1		count	+1:	input
ac2	cur	cur	+2:	output
ac3	link	buf		

Used by a driver process when it is ready for a new operation.
Notice: the procedure delays the process until a relevant message arrives in its queue. Examines the event queue in the following way:

0. answer. Examination continues.
1. message where sender.buf is different from a nonzero reserver.cur: the message is returned by means of send answer (status=illegal, count=0). Examination continues.
2. transput message (operation (15:15)=1) with count=0: the message is returned by means of send answer (status=0, count=0). Examination continues.
3. transput message, where buf.cur equals -1: The message is returned by means of send answer (status=0, count=0). Examination continues.
4. control message (operation (15:15)=0):buf, count and address are saved. Return to control (link+0).
5. input message (operation(14:15)=1):buf, count, and address (mess.2buf) are saved. Return to input (link+1).

6. output message (operation (14:15)=3): buf, count, and address (mess2.buf) are saved. Return (link+2).

Procedure Wait Operation (timer, device, mode, count, buf)

	call:	return:	link:	
ac0	timer	timer	+0 timer	
ac1	device	device	+1 interrupt	
ac2	cur	cur		
ac3	link	cur		
		ac0, ac1 irrelevant	+2: dummy	ac2 cur
ac0		mode		ac3 cur
ac1		count	+3: control	
ac2		cur	+4: input	
ac3		buf	+5:output	

This procedure may be used by a driver process, when it is necessary to wait for either device interrupt, timeout or a message.
buf.cur should contain a value -1, or 0 indicating wait for any buffer, or it should contain a buffer in event queue, in which case a message after this one is waited for.
The timer and interrupt returns are taken if these occur. Dummy return is taken where a message is returned by means of send answer or an answer arrives (see point 0, 1, 2, 3 of NextOperation). The remaining returns are taken when point 4, 5, 6 of Next Operation occurs.

Procedure Set Interrupt (device)

	call:	return:
ac0		destroyed
ac1	device	device
ac2		unchanged
ac3	link	destroyed

Includes the process as user of the device. The device is cleared by a nioc instruction. This means, that any interrupts arriving to the device must be handled by the routine clint.proc. As a standard clint.proc may be: clear. This executes a nioc device.

Procedure Return Answer (status)

	call:	return:
ac0	status	status
ac1	mess2 to buf	destroyed
ac2	cur	cur
ac3	link	destroyed

Insert status a return value for mess2.buf, and the calculated number of transferred bytes into the message buffer (saved buf in optional words). Returns the message buffer to the sender by means of send answer. The number of bytes is calculated by subtracting the original byte address still remaining in the message buffer from the updated byte address saved in the process descriptor.
If one of the clean bits are set in status, buf.cur is set to -1.

Procedure Set Reservation (mode)

	call:	return:
ac0	operation(0:13)	operation (0:12)
ac1		destroyed
ac2	cur	cur
ac3	link	destroyed

If bit 13 of operation (R-bit of mode) is nonzero mess1.buf is examined. If this word is non-zero sender of message is inserted as reserver of cur, otherwise reserver of cur is put to zero.

Procedure Set Conversion (mode)

	call:	return:
ac0	operation(0:12)	operation(0:11)
ac1		destroyed
ac2	cur	cur
ac3	link	destroyed.

If bit 12 of operation (C-bit of mode) is nonzero mess2.buf is inserted as conversion table address.

Procedure Conbyte (byte)

	call:	return:
ac0	byte	byte (converttable.cur + byte)
ac1		destroyed
ac2	cur	cur
ac3	link	destroyed

Loads the byte at relative location byte in conversion table. Note that conversion table address in this case should be a byteaddress. If conversiontable.cur is zero, the procedure is dummy.

Procedure Getbyte (addr, byte)

	call:	return:
ac0		byte addressed
ac1	addr	addr
ac2		cur
ac3	link	destroyed

Fetches the byte at the given byteaddress.

Procedure Putbyte (addr, byte)

	call:	return:
ac0	byte	byte
ac1	addr	addr
ac2		cur
ac3	link	destroyed

Stores the byte, which must be in the range 0 to 255, at the given byteaddress. Note the remaining part of the word addressed is untouched.

I/O HANDLING

The procedures, which can take care of i/o, use zones to describe the activities, with which they are concerned.

They fall into 4 classes, which handle distinct phases of common i/o activities.

Initialisation:

- open
- close
- setposition
- waitzone

Character input/output:

- inchar
- outchar
- outend
- outtext
- outoctal

Record input/output:

- getrec
- putrec

Basic input/output:

- transfer
- waittransfer
- inblock
- outblock

The procedure open readies a zone for actual input/output, and close takes care of orderly closedown of activities. Setposition is mainly of use for block oriented devices.

The character i/o procedures may be used after initialisation and open. They cannot be used with record i/o procedures.

Record i/o procedures may be used after initialisation and open. They cannot be used alongside character i/o procedures. If single bytes of records should be inspected or modified the system utility procedures getbyte and putbyte may be of great help.

Basic i/o procedures are not recommended for general use.

IDENTIFICATION OF A DOCUMENT

The term document is used to describe a medium, which is able to contain data, and which is mounted on a device.

A document is described inside a zone descriptor by:

document name, the process name of the driver, which controls the device.

operation mode, that is the operation, which should be used in any transput operation sent as message to the driver process.

device kind, a word, which contains some bits, that describe how transfer errors may be handled.
At present, the following bits of kind are defined:

b15:	char	: set if the device is character oriented, transfers information in terms of characters
b14:	blocked	: set if a full block should be transfered as a unit.
b13:	positionable	: set if positioning has any effect.
b12:	repeatable	: set if an operation may be repeated

The remaining bits of the kind word should be zero.

Description of mode and kind applicable to standard driver processes, are found as part of their description.

Examples of kinds:

Magnetic Tape Station	1110
Line Printer	0001 or 0011
Card Reader	0010
Teletype	0001
Paper Tape Punch	0001
Paper Tape Reader	0001

RECORD STRUCTURE

There exists three formats for records. For each type, the records may be either blocked or unblocked.

Record type:	Format code:	Blocked:
Unformatted	0	
Fixed length	2	+ 1
Variable length	4	

Unformatted

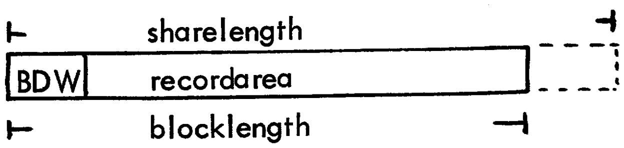
A block contains sharelength bytes or less. In output a full block is transferred to the device regardless of contents. By input as many bytes as requested are delivered from the block. If the records are blocked, change of block takes place, when the remaining bytes of the zone cover the demand insuffiently.

Fixed length

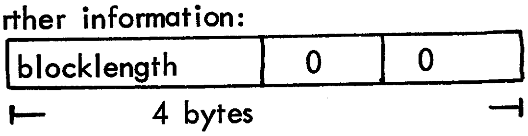
Every block containing one or more records (blocked) of fixed length. The length is given by the zoneparameter reclength. If sharelength is not an integral multiple of recordlength, the last bytes of input are skipped.

Variable length

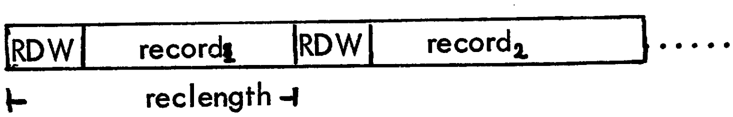
The block contains, in two block descriptor BDW, the length og the total block.



A BDW contains no further information:



The recordarea may contain one (unblocked) or more records. Each record is headed by a 4 byte record descriptor RDW.



A RDW contains the recordlength and a segmentcode, which always is zero.

reclength	0	0

├ 4 bytes ┤

HANDLING OF EXCEPTIONS

In the input/output procedures the user may select certain statusbits, which if set in the answer to a message to the driver, will transfer control to user code. These user facilities are described in the zone descriptor by:

give upmask, giveupaddress.

When the basic procedure waittransfer receives an answer, the statusword is augmented with the following bits:

b15:	repeaterror	is set if the standard repetition of operations has given negative results.
b13:	rejected	is set if a control operation with command = 10_2 is checked.
b12:	position error	, is set if kind (13) is one and filecount or blockcount of answer does not match with the corresponding updated values of the zone descriptor.

This combined driver and standard procedure status is compared with the give-upmask. Common ones from the users_status.

Remaining status bits are given to the standard check actions, which executes the following recovery work:

b0:	disconnected	the error is hard.
b1:	off-line	the error is hard.
b2:	device busy	the operation is repeated.
b3:	- - 1	ignored.
b4:	- - 2	ignored.
b5:	- - 3	ignored.
b6:	illegal	the error is hard.
b7:	eof	the error is hard.

b8:	block_error	the error is hard.
b9:	data_late	if kind (12) is 1 then operation is repeated, otherwise the error is hard.
b10:	parity error	if kind (12) the operation is repeated else it is a hard error.
b11:	end medium	if bytecount of answer is nonzero and operation is input, no action is taken, otherwise the error is hard.
b12:	position error	hard error
b13:	rejected	hard error
b14:	timer	hard error
b15:	repeaterror	hard error

A hard error results in a breakprocess call, with errorcode = 5 and status placed in ac1.

An operation is repeated a maximum of 5 times. If it is still erroneous, it is classified as having a repeaterror. The cause of the unsuccesful repeats is included in user status.
When remaining bits have been treated by the standard actions, control is given to giveupaddress if users_bits are different from zero. Otherwise a normal return from wait_transfer takes place.

Exit to giveupaddress takes place with:

ac1:	users bits of status.
ac2:	zone
ac3:	return address
ztop:	first byte transferred.
zrem:	actual bytecount for transfer.
z0:	user bits of status.

The giveup action may return to . repeatshare or directly to ac3 fo call. ac3 and ac2 must be unchanged in either case.

If the giveup action returns to .repeatshare the message to the driver is repeated. If the giveup action returns directly to ac3 the answer is treated as correct and control is returned to the calling I/O procedure.

Note: The giveup action must never call any I/O effecting procedure if it wants to return to the calling I/O procedure by means of .repeatshare or via ac3.

FORMATS

Zone

A zone describes an input/output situation for a process. It consists of a zone descriptor and a buffer.

The zone descriptor contains general parameters. The buffer contains the share descriptors and the shares.

Zone Descriptor

A zone descriptor is identified by the address of its first location.

zname.zone: document name (three words).

The document name of one to five characters identifies the driver process which should receive messages with i/o requests.

zmode.zone: operation.

This operation is used in transput messages to the documents.

zkind.zone: kind of document.

Kind for error handling, open action, close action, etc.

zmask.zone: mask for give up.

The mask is compared with the status word when a transfer is checked. Common ones form the users bits and causes the address for give up to be entered.

zgive.zone: give up address.

This address is entered if users bits is non-zero.

zfile.zone: file count.

Used for positioning of some document kinds.

zbloc.zone: block count.

Used for positioning of some document kinds.

zconv.zone: conversion table address.

Used in control message to driver process from open.

zform.zone: format code for records,

zleng.zone: length of records,

zfirs.zone: first of record (byte address).

Address of the first byte in the record.

ztop.zone: top of record (byte address)

Address of the first byte after the record.

zused.zone: used share

Address of the currently used share.

zshar.zone: share length (in bytes).

All shares have the same length.

zrem.zone: remaining bytes in share.

The bytes represent already input characters or room for new output records.

The zone contains a number of auxiliary words, used by the procedures. The number of these are given by the assembly constant zaux. These are labelled z0, z1, ...z"aux-1".
The total size of a standard zone descriptor is given by the field z.

Share Descriptor

A share descriptor is identified by the address of its first location.

soper.share: operation (0.message)

scoun.share: count (1.message)

saddr.share: address (2.message)

sspec.share: special (3.message)

These first four words are used as message to the document.

snext.share: next share.

Next share descriptor in the linked cyclical list of share descriptors in the zone.

sstat.share: state of share with the values:

0	free
buf	pending

sfirs.share: first shared (byte address).

Address of first location in the share, always even.

The size of a share descriptor is given by ssize.

The total used core for a zone with N shares of length B is:

$$z + N * (ssize + (B+1)//2)$$

BASIC I/O PROCEDURES

procedure Transfer (zone, length, operation);

	call:	return:
ac0	operation	destroyed
ac1	length	destroyed
ac2	zone	zone
ac3	link	destroyed

Initiates a transfer operation described by operation to used_share.zone. The bytecount of the message is put to length. Sharestate of used share points to the buffer used for the message. Used share is updated to next share.

Note: starttransfer does not check that state of used share is free (zero). If the state is not free, the buffer address saved in state is lost permanently.

procedure Wait transfer (zone)

	call:	return:
ac0		destroyed
ac1		destroyed
ac2	zone	zone
ac3	link	destroyed

Examines usedshare.zone. If state is free (zero) the procedure returns immediately, otherwise it waits for answer to the message placed in buffer identical with state and sets state to free.

When the answer arrives the status is checked as described in HANDLING OF EXCEPTIONS. top.zone is adjusted to point at firstaddress of share. remaining.zone is adjusted to bytecount of answer.

procedure Inblock (zone)

	call:	return:
ac0		destroyed
ac1		destroyed
ac2	zone	zone
ac3	link	destroyed

Administrates the basic cyclic buffering strategy for input procedures as inchar or getrec. The algorithm is:

```
while state.usedshare.zone = 0 do
    transfer (zone, sharelength.zone, mode.zone);
comment zone should be opened for input;
wait_transfer(zone);
comment n-1 shares are busy and one is
    ready with input;
```

procedure Outblock (zone)

	call:	return:
ac0		destroyed
ac1		destroyed
ac2	zone	zone
ac3	link	destroyed

Administrates the basic cyclic buffering of output. The algorithm is:

```
transfer (zone, sharelength.zone - rem.zone, mode.zone);
comment zone should be opened for output;
waittransfer(zone);
remaining.zone:= sharelength.zone;
```

INITIALISATION PROCEDURES

Procedure Open (zone, operation)

	call:	return:
ac0	operation	destroyed
ac1		destroyed
ac2	zone	zone
ac3	link	destroyed

The operation is placed in the modeword of zonedescriptor.
Then remaining bytes of zone is initialized to zero if command = 1 or else to sharelength. Top of zone points to first of used share.

To initialize the transfers a control message is sent to the process specified by zname. This message includes reservation and set up of conversion table address.

Procedure Setposition (zone, file, block)

	call:	return:
ac0	block	destroyed
ac1	file	destroyed
ac2	zone	zone
ac3	link	destroyed.

Waits for all pending transfers to the zone as described in procedure close.
Then it sends a position message, which contains the new file- and blockcount

Procedure Close (zone, release)

	call:	return:
ac0		destroyed
ac1	release	destroyed
ac2	zone	zone
ac3	link	destroyed

First the zone is set neutral by means of waitzone.

Then if command was output, a termination, and if release is nonzero a release reservation, disconnection control message is sent to the process specified by name of zone.

Command is set to zero, and the zone is set neutral by waitzone.

Procedure Waitzone (zone):

	call:	return:
ac0		unchanged
ac1		unchanged
ac2	zone	zone
ac3	link	destroyed

Terminates the current activities of the zone as follows:
If command is output, a last block is output.
Then all pending transfers are waited for. Either by means of wait transfer if command is output, or else by means of waitanswer (no checking takes place).

CHARACTER I/O PROCEDURES

Procedure Inchar (zone, char)

	call:	return:
ac0		
ac1		char
ac2	zone	zone
ac3	link	destroyed

Gets the next 8-bit character from the zone.

Procedure Backspace (zone)

	call:	return:
ac0		destroyed
ac1		top.zone
ac2	zone	zone
ac3	link	link

Delivers the latest character read by inchar from the zone. Consecutive calls have the same effect as one call.

Procedure Outchar (zone, char)

	call:	return:
ac0		unchanged
ac1	char	destroyed
ac2	zone	zone
ac3	link	destroyed

Procedure Outend (zone,char)

	call:	return:
ac0		destroyed
ac1	char	destroyed
ac2	zone	zone
ac3	link	destroyed

Outputs the 8-bit character on the zone by means of outchar. Then outputs the part of the share now filled with characters by means of outblock. This output of the last portion is done only for character oriented devices, i.e. kind(15) = 1.

Procedure Outtext (zone, address)

	call:	return:
ac0	address	destroyed
ac1		destroyed
ac2	zone	zone
ac3	link	destroyed

Outputs the text of 8-bit characters on the zone by means of outchar. Address is a byte address and may be odd as well as even. The text terminates with a zero character.

Procedure Outoctal (zone value)

	call:	return:
ac0	value	destroyed
ac1		destroyed
ac2	zone	zone
ac3	link	destroyed

Converts the value to character form and outputs it on the zone by means of outchar. The 16-bit value is output as 6 octal digits.

RECORD I/O PROCEDURES

Procedure Getrec (zone, addr, bytes)

	call:	return:
ac0	(bytes)	bytes
ac1		addr (first byte of record)
ac2	zone	zone
ac3	link	destroyed

Makes the next record available in inputbuffer. Depending on recordformat the actions are:

Unformatted unblocked:

```
inblock (zone);
bytes: = rem.zone;
goto update;
```

Fixed length unblocked:

```
inblock (zone);
bytes: = length.zone;
comment recordlength is used;
goto update;
```

Unformatted blocked:

```
length.zone: = bytes;
```

Fixed length blocked:

```
bytes: = length.zone;
if rem.zone < bytes then inblock (zone);
goto update;
```

Variable length blocked:

```
if rem.zone > 0 then goto next-record;
```

Variable length unblocked:

```
inblock (zone)
rem.zone: = top.zone (0:1) -4;
top.zone: = top.zone +4
```

```
next-record:
                    bytes: = top.zone (0:1) -4;
                    rem.zone: = rem.zone -4;
                    top.zone: = top.zone +4;
update:             if bytes < rem.zone then
                    breakprocess (cur,4);
                    addr: = first.zone: = top.zone;
                    top.zone: = top.zone + bytes;
                    rem.zone: = rem.zone -bytes
                    length.zone: = bytes ;
```

Procedure Putrec (zone, addr, bytes)

	call:	return:
ac0	bytes	destroyed
ac1		destroyed
ac2	zone	zone
ac3	link	destroyed

Makes space for a record in the output buffer. Depending on recordformat the actions are:

Fixed length unblocked:

```
bytes:= length.zone;
```

Unformatted-unblocked:

```
outblock (zone);
if rem.zone < bytes then
  breakprocess (cur, 4);
update top of zone and rem of zone;
return;
```

Fixed length blocked;

```
bytes:= length.zone;
```

Unformatted blocked:

```
if rem.zone < bytes then outblock (zone);
if rem.zone < bytes then
  breakprocess (cur, 4);
update top of zone and rem of zone;
return;
```

Variable length blocked:

```
if rem.zone < bytes + 4 then
  outblock (zone);
```

Variable length unblocked:

```
outblock (zone);
if rem.zone < bytes + 4 then
   breakprocess (cur, 4);
top.zone (0:1):= bytes + 4;
top.zone (2:3):= 0;
top.zone:= top.zone + 4;
rem.zone:= rem.zone - 4;
update top of zone and rem of zone;
first.used.zone (0:1):= sharelength.zone - rem.zone;
first.used.zone (2:3):= 0;
comment block descriptor words inserted;
return;
```

Update top of zone and rem of zone:

```
first.zone:= top.zone;
top.zone:= top.zone + bytes;
rem.zone:= rem.zone - bytes;
```

Errors:

The process is breaked with errornumber = 4, if an improper number of bytes are specified.

Procedure Move (paramaddr) ;

	call:	return:
ac0		destroyed
ac1		destroyed
ac2	paramaddr	paramaddr
ac3	link	destroyed

paramaddr	+0	count
	+1	to_address
	+2	from_address
	+3	work location

The procedure moves count bytes from byte position from_address and on to byte position to_address and on.

Note: The procedure always moves to full words. If to_address + count is odd one more byte is destroyed.

OPERATOR PROCESS

General Rules

An operator process coordinates the communication with the operator.
The process cannot be reserved.
The process works on an input device and on output device.

Attention Request

If output is in progress this will continue until End-of-Line, and the rest is skipped.
Input in progress is cancelled.
A line is read from the input device and interpreted like this:
The line contains a name. The operator process searches in its event queue for an input message from a process with this name. The action depends on the name:
If the name is not a provess name, the following line is output on the output device:

unknown

If the event queue of the operator process contains no message from the process with the found name, the following line is output on the output device:

busy

If the event queue of the operator process contains at least one message from the process with the found name, the process is selected as current process, and the message is executed.

Control Message

The message is returned (status = 0).

Input Message

Input messages will be delayed until operator enters a character.
The action depends on the sender:

1: If sender = current process, the message is executed, that is a line is read from the operator input device.

2: Sender ≠ current process.
The message buffer is left unchanged in the event queue and may only be executed after an attention message.

Output Message

If input is in progress this will continue until End-of-Message or until Timeout occurs.
The action depends on sender:

1. Sender = current process.
The text is output on operator output device.

2. Sender ≠ current process.

2.1 The sender is selcted as current process. The text
> "proc" is output followed by the text of the message.

Messages and Answers

Operation:	Message:	Answer:
control	0	0
input	1	0
	bytes	bytes input
	address	
output	3	0
	bytes	bytes output
	text address	

Operating System S

The operating system S contains basically a command line interpreter, which is able to execute system altering commands.

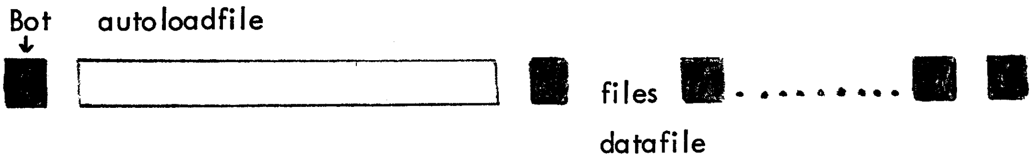
S has always two sources of input:

1) Primary input, which is fixed at system generation time.
2) Normal input, which is alterable by execution of commands.

S has furthermore one output device, which is fixed at system generation time.
Commands consists of sequences of ASCII texts seperated by spaces and terminated by one control character (ASCII 0-31).

Files are identified by an ASCII text in the first block. Any characters betwen the terminating linefeed and data should be blanks (Null).

MT program tape:



Datafile:

Commands

IN:	"device"	selects "device" as normal input. If "device" is not a process the error message: UNKNOWN will be output.
INT:	{"ident"} $_{0}^{1}$	reads a sequence of command lines from normal input. The command lines should contain END as last command. "ident" is used to identify the file from which input should commence.
END:		dummy command.
START:	"procname"	starts a process, i.e. admits it to continue if it is stopped. If the process is not found error message UNKNOWN is given.
KILL:	"procname"	removes a process from chain. If the program of the process is specified as own, it is also removed. If the process is not found error message UNKNOWN is given.
LIST:		lists all existing processes, and the current maximal load address.
STOP:	"procname"	stops a process, i.e. prevents it from entering running queue. If it is not found error message UNKNOWN is given.
BREAK:	"procname"	starts a process in its breakaddress. If it is not found error message UNKNOWN is given.
CLEAR:		acts as a sequence of KILL commands on all user processes.

LOAD: {"ident"}$_0^*$

1) if any of the given idents are found as processes they are removed from the parameter list.

2) then the normal input device is searched for files identified with "ident"s. If any are found they are loaded in relocatable format.

If the parameter list is empty loading starts at once from normal input.

Errormessages:

SUM appears if a relocatable block contains a checksum error.

ILLEGAL appears if a relocatable block contains no proper startcode.

"device" XXXXX appears in case of a input device status. Answering with START means that execution will continue.

STANDARD DRIVER PROCESSES.

TELETYPE CONSOLE DRIVER

General Rules

This is an operator driver.

Control messages are ignored.

The driver operates on bytes placed left to right in words, allowing odd addresses and odd byte counts.

No parity check is performed: the parity bits are masked off.

Control

Ignored

Input Operation

A line of characters can be input to the storage area defined by the message.

The characters are echoed on the teletype during input.

Some characters have special effects.

RUBOUT:	Char cancel The character " ← " is output on the teletype and the last character is cancelled in the storage area.
RETURN:	New line. A LF character is stored. An answer is delivered with status = 0 and bytes input defined.
BELL: (© G)	Attention Request. Previous input is forgotten.
LINE FEED	Same action as for RETURN.

Input is terminated when the storage area is full, or when one of the special character actions causes termination, whichever occurs first.

Input is terminated temporarily when no character has been input for 5 seconds. If an output message is pending the input is cancelled, otherwise input is continued.

Attention Request

Attention request is set when any character key is activated in a non-input situation, or when the BELL key is operated. In a output situation output beyond the present line is skipped.

Output Operation

The storage area defined by the message is output as 7-bit characters.
Output is terminated when the area is empty, or after a NULL character.

Status

Not applicable.

PAPER TAPE READER DRIVER

General Description

Control- and Input messages are accepted. Output messages are interpreted as input messages. Leading blanks are skipped after a control message.

Control

Reservation and Conversion is accepted. Conversion table address should be an even byte address.

Input Operation

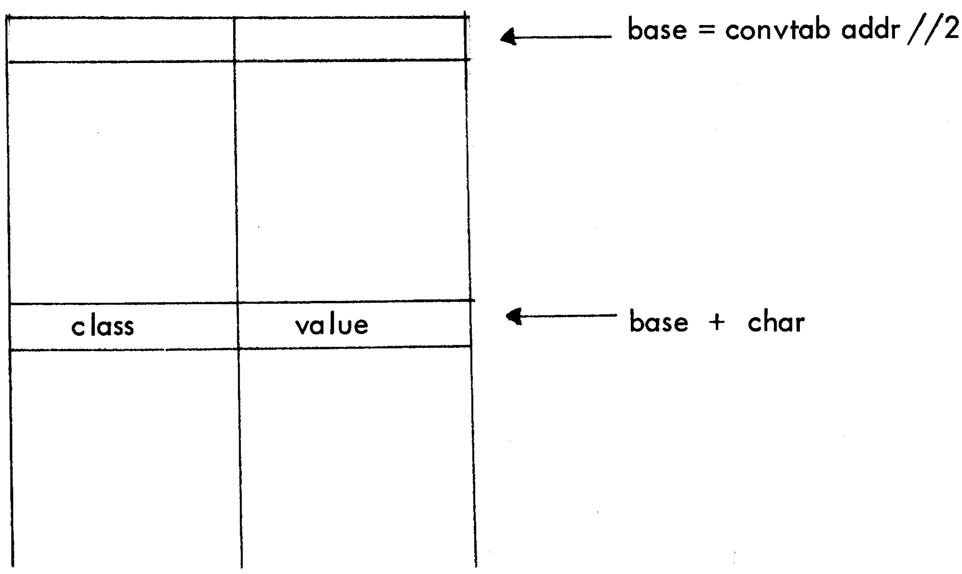
Three modes of operation exist.

1:	binary, the input character is delivered.
5:	odd parity, the most significant bit is regarded as a complemented parity and removed.
9:	even parity, the most significant bit is regarded as parity and removed.

When operation 5 or 9 are used, the further treatment of the input character is:

```
char: = dia (device);
if ¬ parity_ok (char) then char: = 26;
char: = char mod 128;
```

Conversion is done with the following table:



class = 0	normal character,	the value is delivered.
class = 128	skip character,	the character is skipped.
class = 129	Shift in,	base: = base - if value = 0 then 256 else value. The character is looked up again.
class = 130	shift out,	base: = base + if value = 0 then 256 else value. The character is looked up again.
class = 132	end of block,	the character delimits the current block.
class >= 136	parity	parity error indicated and counted; value delivered.

Status

1B3:	Char defect.
1B6:	Device reserved.
1B10:	Parity error is set if one or more characters with wrong parity occurs in the block.
1B11:	End medium is set if the device is not ready to deliver a character within 1 second.

PAPER TAPE PUNCH DRIVER

General Description

Control- and Output messages are accepted. Input messages are interpreted as control messages.

Control

Reservation and Conversion is accepted. Conversion table address should be a byte address, and the characters are converted before output by simple indexing.

conv character: = byte (convtable + character).

Position, Termination and Disconnect results in output of 128 characters blank tape.

Output operation

Three modes of operation exist:

3:	binary, the converted character is output.
7:	odd parity, the converted character is augmented by the complement of its parity in the most significant position.
11:	even parity.

Status

1B6:	device reserved
1B11:	em, is set if the hardware status bit 15 (paper low) is set.
1B14:	timer, is set if a character is not output within 600 ms.

LINE PRINTER DRIVER

General Description

Control- and Output messages are accepted. Input messages are treated as Control messages.

Control

Reservation and Conversion is accepted.

Conversion table address should be a byte address. The characters to be output are converted as

conv char:= byte (convtable + char);

The position command has only effect if Output mode is 7. Then the mess2 part (file) of the control message gives a number of spaces to be output in front of the printline.

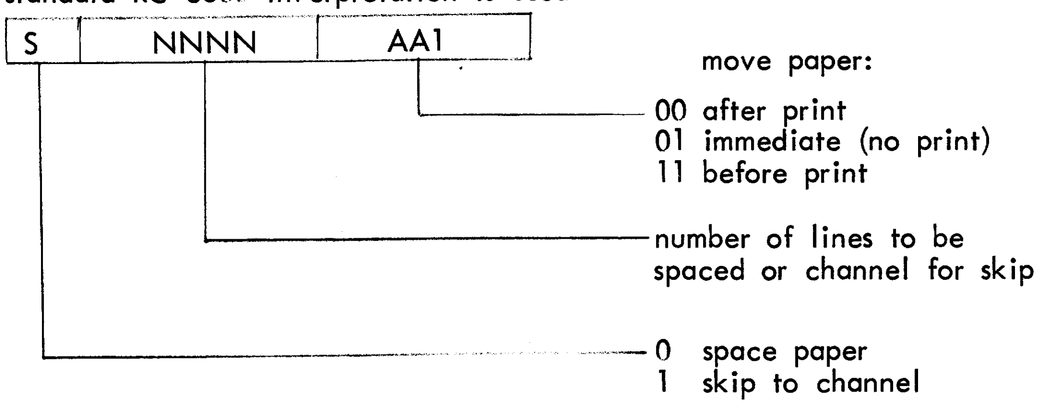
Output Operation

Two modes of operation exist:

3 : The converted characters are output to the line printer.

7 : The first byte of an output message is interpreted as a carriage control word.

The standard RC 3600 interpretation is used



Status:

1B0:	disconnected, line printer disconnected or unable to accept command.
1B1:	off-line, the line printer is in off-line state.
1B3:	device mode 1, channel 12 in carriage control tape has been encountered.
1B6:	illegal, device reserved.
1B8:	block error, paper is out, paper torn or paper run away.
1B9:	data late, line printer not in ready state.
1B10:	parity, a ccw contains a zero bit in the least significant position.
1B11:	end medium, end of paper, less than 1.5 forms left.
1B14:	timer paper run away.

MAGTAPE DRIVER

General Description

Control, Input- and Output messages are accepted. At return mess2 and mess3 of the message contains the actual file- and block number.

Control

Reservation, Termination, Positioning, Disconnect, Erase and Sense are accepted. Conversion is ignored.

Bit 11 = 1 :	Termination:	Writes two tape marks and positions between them.
Bit 10 = 1 :	Positioning:	Positions the tape at the file- and block number given by mess2 and mess3 of the message.
Bit 9 = 1 :	Disconnect:	Rewinds the tape and when BOT is sensed, sets the unit off-line.
Bit 8 = 1 :	Erase:	Erase 3.7 inches of the tape.

Input

Operation:

1 :	Read packed, byte limit = 18.
5 :	Read packed, byte limit = 0.
9 :	Read unpacked, byte limit = 18.
13 :	Read unpacked, byte limit = 0.

Output

Operation:

3 :	Write.

Common Input/Output

Common mode bits:

Bit 4 = + 8192:	Selects the lower of two possible densities.
Bit 5 = + 4096:	Selects even parity.

<u>Status:</u>

1B1 :	off-line, the unit is off-line.
1B2 :	Busy ; the unit is rewinding.
1B3 :	device mode 1, noise record.
1B4 :	device mode 2, PE.
1B5 :	device mode 3, write lock.
1B6 :	illegal ; byte limit conflict or write lock, device reserved.
1B7 :	eof, end of file.
1B8 :	block error, block length error (read).
1B9 :	data late.
1B10:	parity error, read or write.
1B11:	end medium, end of tape (EOT).
1B12:	position error, a file mark is read while positioning at a block. The blockno of answer is one greater than number of blocks in current file. Position in front of filemark.
1B14:	timer, time out at wait interrupt.

CARD READER DRIVER

General Description

Control and Input messages are accepted. Output messages are returned with illegal status.

Control

Only Control modes: Reservation and Conversion are performed, other modes are skipped. A sense of the device is always executed when the driver receives a control message.

Input Operations

Three modes of operation exist:

1:	Read binary bytes.
5 :	Read binary punched cards.
21 :	Read decimal punched cards.
33 :	Read decimal punched cards and skip of trailing blank columns. (Min. 10 columns are returned)

All other operation modes are returned with illegal status.

Buffer start	B	C (1)
	B + 1	C (2)
	B + 2	C (3)
		C (n-1)
	B+n-1	C (n)

Fig. 1 Column lay-out in internal driver buffer area
Number of columns = n.

Binary Read Mode

The binary read command will cause the card to be transferred to the internal buffer as shown in fig. 1. Fig. 2 shows how the row numbers in each column are placed in the buffer.

Not used				12	11	0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Fig. 2 Buffer word

Decimal Read Mode

The decimal read command will cause the card to be transferred to the internal buffer as shown in fig. 1 Fig 3 shows how the 12 bits in a column will be converted to a 8 bit character.

E								12	11	0	8	9	x	x	x
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Fig. 3 Buffer word

The table below shows how the E-bit (error) and the three x bits are converted.

Punch position number							Buffer word bit			
1	2	3	4	5	6	7	0	13	14	15
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	1	0	1
0	0	0	0	0	1	0	0	1	1	0
0	0	0	0	0	0	1	0	1	1	1
More than one 1's							1	x	x	x

Data - conversion

The Conversion Table Address should be a byte address. If the operation is 5 (Read binary punched cards) no conversion of data is performed and the card columns (fig 2) is delivered to the user. E.g. one card column fills up one user buffer word.

If the operation is 21 or 33 (Read decimal punched cards) the characters are converted before output by simple indexing:

conv character: = byte (conv table + character)

E.g. one card column fills up one user buffer byte.

Status

1B1 :	off-line, card reader is not ready: Stacker jam, Stacker full, Stop five, power off, test mode, open interlock, start true, disconnected.
1B2 :	Busy, a card has failed to feed from hopper, a card has not passed over the read station in the specified time.
1B3 :	Device mode 1, reject command failed.
1B4 :	Device mode 2, 51 column cards.
1B6 :	Illegal command, device reserved.
1B8 :	Block length error, card reader has delivered less than 80 columns, user byte count less than received no. of bytes from card reader.
1B9 :	Data late.
1B10:	Parity error, light senser failure, at least one column in a card read in decimal mode contains a conversion error.
1B11:	End medium, hopper empty (no data has been transferred).
1B14:	Timer, in wait interrupt has occurred (300 msec). The received data are delivered.

OPERATOR CONTROL PANEL DRIVER.

General Description and Rules.

This is an operator driver. The driver operates on bytes placed left to right in words. Odd addresses are not allowed but odd byte counts are.

Attention Request.

When the load button is pressed, the driver accepts this as a request to choose the operating system 's' as user. All input is then directed to 's' until 'NL' button or 'CAN' button is pressed. In attention mode the lamp labelled 'LOAD' is lit.

Control.

Control messages are ignored.

Input Operation.

A string of digits or an indicator message can be input to the storage area defined by the message. Only the digit, +, - buttons are echoed from position 12 on the display.

The other indicators have following effects:

CAN:	Echoed characters are cancelled and previous input is ignored. The operator can input a new text.
NL:	Terminates the previous input message by placing a NL as the last character of the string of digits.
START, CONT, STOP, INT:	(1) If an input operation is in progress from keyboard it terminates the previous input message by placing a NL as the last character of the string of digits. The button is then treated as for (2) if a new input message arrives.

START, CONT, STOP, INT: (2) If no previous input has occured the corresponding text: 'START'. 'CONT', etc. is delivered as answer.

Note: If more than one indicator is activated the left most is delivered.
On Keyboard in sequence: CAN,NL,SP,+,-, 0,1,...,8,9.

Input is terminated when the storage area is full, or as explained above, whichever occurs first.

An answer is delivered with status = 0 and bytes input defined.

Output Operation.

$b_7b_6b_5$

$b_4b_3b_2b_1$	000	001	010	011	100	101	110	111
0000	terminator		SP	Ø	@	P		
0001	terminator		!	1	A	Q		
0010	terminator		"	2	B	R		
0011	CLEAR ALL		#	3	C	S		
0100	START		$	4	D	T		
0101	CONT		%	5	E	U		
0110	STOP		&	6	F	V		
0111	ALARM		'	7	G	W		
1000	C.STOP		(	8	H	X		
1001	C.ALARM		)	9	I	Y		
1010	NL		*	:	J	Z		
1011			+	;	K	[		
1100			,	<	L	\		
1101			-	=	M	]		
1110	C.START		.	>	N	{		
1111	C.CONT		/	?	O	}		

The figure shows the character repertoire and coding (binary). Empty positions indicate blind characters.

Output is delimited by the bytecount or by a character < 3.

Outputting the NL character causes all 16 character-positions in display to be cleared in one operation.

Displayable characters delivered after the NL character are displayed in sequence from left to right. If more than 16 displayable characters are delivered after the last NL character, the previous message is overwritten cyclically.

Blind characters have no effects.

The characters 'CLEAR ALL' to 'C.CONT' on figure are used to indicate the corresponding function indicators:

<3> means clear all lamps and audio alarm.
<4> means light the lamp labelled 'START'
<5> - - - - - 'CONT'
<6> - - - - - 'STOP'
<7> - turn the audio alarm on
<8> - clear the lamp labelled 'STOP'
<9> - turn the audio alarm off
<10> - NL
<14> - clear the lamp labelled 'START'
<15> - - - - - 'CONT'

Note: Depressing any indicator turns the audio alarm off.

Status.

Not applicable.

Reader Punch Driver.

General Description.

Control, Input and Output messages are accepted.

Control.

Only Control modes, Reservation and Conversion are performed, other modes are skipped. A sense of the device is always executed when the driver receives a control message.

Input/Output Operations.

General.

In the Input/Output Operations bit (7 : 15) are used to define the type of Command.

Bit 7 = 1 selects stacker # 2 or the secondary hopper depending on the mode of operation

Bit 8 = 1 inhibits the input feed of the next card

Bit (9 : 15) defines the mode of operation

The driver communicates with the device in either binary or decimal mode.

Binary mode.

In binary mode the driver words correspond to the card columns as shown in fig. 1.

Not used		12	11	0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Fig. 1. Driver word.

Decimal mode.

In decimal read mode the card column will be transferred to the driver as an eigth bit character as shown in fig. 2.

In decimal write mode the driver word will be transferred to the device as shown in fig. 3.

Fig. 4 shows the correspondence between bit(13 : 15) in the driver word and row (1 : 7) in the card column, and for the decimal read mode how the E-bit (Error) is converted.

E								12	11	0	8	9	x	x	x
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Fig. 2. Driver word in decimal read mode.

								12	11	0	8	9	x	x	x
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Fig. 3. Driver word in decimal write mode.

Card row number							Driver word bit			
1	2	3	4	5	6	7	0	13	14	15
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	1	0	1
0	0	0	0	0	1	0	0	1	1	0
0	0	0	0	0	0	1	0	1	1	1
More than one 1's							1	x	x	x

Fig. 4. Data Conversion in decimal mode
(the x's are irrelevant).

Input Operations.

Bit 7 = 1 Selects the secondary hopper.

Bit 9 = 1 denotes that the card in the wait station will be led out to stacker # 2.
A new card is read from the specified hopper.

Bit (10 : 15) denotes the mode of operation.

Four modes of operation exist:

1: Read binary byte

5: Read decimal byte and convert

9: Read binary word

21: Read decimal byte, convert and skip trailing blank columns (min. 10 columns are returned).

All other modes of operation are returned with illegal status.

Except for the abovementioned case (bit 9 = 1) stacker # 1 is used.

In the decimal modes and in read binary byte mode, byte count must be greater than or equal to 80 (51), or else the operation is returned with block length error and a number of bytes transferred equal to block count. The rest of the columns will be skipped.

In the read binary word mode, byte count must be greater than or equal to 160 (102), or else the operation is returned with block length error, and a number of words equal to byte count // 2. The rest of the columns will be skipped. Byte count and byte address must be even numbers or else the operation is returned with illegal status.

Output Operation.

Bit 7 = 1: Selects stacker # 2.

Bit 9 = 1: The card in the wait station is led out to the appropriate stacker, and a card is fed before the write operation.

Bit (10 : 15) denotes the mode of operation.

Five modes of operation exist:

- 3 : Punch binary byte with conversion.
- 11 : Punch decimal byte with conversion.
- 19 : Print decimal byte with conversion.
- 27 : Punch and Print decimal byte with conversion.
- 47 : Punch binary word.

All other operation modes are returned with illegal status.

If the punch/print modes the secondary hopper is used.

In the decimal modes and the punch binary byte mode, byte count must be less than or equal to 80 (51). If not, the operation is returned with block length error, and 80 (51) bytes will be transferred.

In the punch binary word mode byte count and byte address must be even numbers, or else the operation is returned with illegal status. Byte count must be less than or equal to 160 (1o2), or else the operation will be returned with block length error and 80 (51) words transferred.

In all modes if byte count is less than the maximum allowed, the card will be filled up with trailing blanks.

Data Conversion.

The Conversion Table Address should be a byte address.

If the operation is 9 (read binary word) or 47 (punch binary word) no conversion

of data is performed and the twelve last bits in a user word corresponds to a card column (fig. 1).
If the operation is 1, 5, 11, 19, 21, 27, or 43 (read/write decimal or binary byte) the characters are converted before output/input by simple indexing:

conv char : = byte (conv. table + char)

e.g. one card column corresponds to one user buffer byte.
To avoid conversion the Conversion Table Address should be zero.

Status.

1B1 : Off-line, Reader Punch is not ready:
Power off, power start-up, open interlock, Input Error, Output check, Punch Data Check, Stacker Full, Stop/Reset key was pressed, disconnected.

1B2 : Busy, a card has not reached the read station in a specified time, or a light senser failure.

1B3 : Device mode 1, Stacker Full.

1B4 : Device mode 2, 51 column cards.

1B6 : Illegal command.

1B7 : End of file, secondary hopper empty.

1B8 : Block length error, user byte count does not comply with the number of columns in a card.

1B10 : Data error, read error, punch error, at least one column in a card read in decimal mode contains a conversion error.

1B11 : End medium, primary hopper empty.

1B14 : Timer, in wait interrupt has occurred. The received data are delivered.

CASSETTE TAPE DRIVER.

General.

Control, Input- and Output messages are accepted. At return mess 2 and mess 3 of the message contains the actual file- and block number.

Control.

Reservation, Termination, Positioning, Disconnect, Erase and Sense are accepted. Conversion is ignored.

Bit 11 = 1 : Termination; writes two tape marks (according ECMA 34 vers 2) and positions between them.

Bit 10 = 1 : Positioning; positions the tape at the file- and block number given by mess 2 and mess 3 of the message.

Bit 9 = 1 : Disconnect; rewinds the tape and ejects the cassette.

Bit 8 = 1 : Erase; erases the tape in a length corresponding to approximately 150 characters.

Input

As it is possible to read data according several standards, following input modes are valid:

Operation:

1: Read one block, ECMA 34 vers 2.
9: Read one block, ECMA 34 vers 1.
17: Read one block, ASCII Standard (LRC + VRC).
33: Read one block, no check of data.

Read continously mode: If 4 is added to the abovementioned input modes, the CTU will read the datablocks without stopping at interblockgabs, if input messages are contiguous within 20-40 ms.

Output.

Operation:

3: Write, according ECMA 34 vers 2.

Status:

1B1 :	Off-line, disconnected, cassette released.
1B2 :	Not ready, busy, rewinding or positioning to BOT.
1B3 :	Device mode 1: End of Data, erased tape with length both of 250 to 400 mm has been forund.
1B4 :	Device mode 2: Even parity on character was found, when input mode 17 is used. Substitute char 26 is inserted.
1B5 :	Device mode 3: Power interrupt, is indicated when power failure and after power on.
1B6 :	Illegal, illegal command.
1B7 :	EOF, end of file.
1B8 :	Block length error (read).
1B10 :	Parity or blockcheck error.
1B11 :	End Medium, BOT/EOT hole is sensed in the tape during operation (data is not lost).
1B14 :	Timeout; possibly hard error in cassette tape unit, or writing has been tried on a cassette with the write-enable plugs removed.

DISC DRIVER

General Description

Control, Input, and Output messages are accepted. The driver can handle up to 4 disc drives, connected to the same controller.

Each disc pack mounted on a drive contains 2 surfaces of 203 cylinders of 12 sectors. Giving a total of 2 x 203*12 = 4872 sectors of 512 bytes.

Addressing of drives and sectors uses a logical segmentnumber. Where the physical addresses are computed as:

driveno	=	logical segment / 4872
cylinder	=	logical segment / 24 mod 203
surface	=	logical segment / 12 mod 2
sector	=	logical segment mod 12

Control

Reservation, and position are accepted. Other control messages are ignored.

Position uses blockno = mess3.buf as current logical segmentnumber, and positions the disc accordingly.

Input

Two modes of operation exist.

1 sequential read

9 random read

The bytecount of message shall be an integral multiple of 512. Address of message shall be an even byteaddress.

In sequential read mode bytecount / 512 sectors are read from current segment and forward. Increasing current segment by one for each sector read.

In random read mode current segment is first assigned the value of mess3.buf, and then input takes place as for sequential read.

Note: When using the standard I/O system sharelength should be exactly 512. Otherwise repetition will function incorrectly.

Output

Two modes of operation exist:

3 write sequential
11 write random

The bytecount of message shall be an integral multiple of 512. Address of message shall be an even byteaddress.

In sequential write mode bytecount / 512 sectors are written from current segment and forward. Current segment is increased by one for each sector.

In random write mode current segment takes the value of mess3.buf, and output is then performed as in sequential mode.

Note: When using the standard I/O system sharelength should be exactly 512, otherwise repetition will function incorrectly.

Answers

In any answer from the driver mess1 gives the number of bytes transferred. Mess3 contains current logical segment number.

Status

1B0	disconnected	the selected drive is not available.
1B6	illegal	the disc driver is reserved, address is an odd byteaddress. Bit 12 drive malfunction of hardware.
1B8	blockerror	the bytecount is not an integral multiple of 512 bytes.
1B9	data late	data channel overrun
1B10	parity	parity error on disc sector in read operation.
1B11	end medium	drive does not exist, end of cylinder detected without zero sector counter (hardware bit 11)
1B12	position error	a seek has failed (hardware bit 10)
1B14	timeout	an operation has not terminated within 200 ms (seek, I/O) or 640 ms for recalibrate.

MULTIPLEXER DRIVER

GENERAL DESCRIPTION

Control, input- and output messages are accepted. The driver exists in 3 versions: one for 16 half duplex channels, one for 32 half duplex channels and one for 64 half duplex channel.

Initialization

A message of the format:

mess0 : -1
mess1 : count
mess2 : address
mess3 : irrelevant

is used to initialize the line descriptions. The lines are numbered 0 to N (N = 15, 31, 63) logically and address and count points to a block of words giving the type and physical channel-number of each line.

```
address/2 :   line descriptor 0  ┐
                     .           │
                     .           │
                     .           ├   "count" bytes.
                     .           │
                     .           │
               line descriptor N ┘
```

Where each line description occupies 4 bytes

echo	channel, kind
time0	time1

echo is the echo line for a full duplex input line, otherwise 255.
Channel, kind is the physical channel number * 2 + 1 if output or combined input/output channel.

Time0 is only relevant for input channels, and is a maximal wait time for first character (in full seconds) 0 denotes no max. wait time.

Time1 is the allowable wait time between characters (in full seconds).

Line addressing

For all normal messages, the line number is given in the left byte of mess0 (operation).

CONTROL

Reservation, conversion and disconnect are accepted. Termination, Position and Erase have no effect.

operation (0 : 7) = line

Reservation: The line is reserved/released depending on mess1.
In case of reservation of an output or half duplex channel, a SET DATA TERMINAL READY command is given.

Conversion: Mess2 is taken as a byte-address of an conversion table. This table should start at an even byte (Full word boundary).

Disconnect: A HALT command is executed, and in case of output and half duplex line a CLEAR DATA TERMINAL READY. Attention buffers are returned.

INPUT

Operation (0 : 7) = line

The remaining part of operation, selects the input buffer to be used as an attention buffer, or a normal input buffer. An attention buffer is retained until no normal input buffer is present, and an attention character arrives.

Operation (8 : 15):

1	normal input
17	attention
+4	parity check for even parity of characters.
+8	echoing of input takes place to echo channel.

Treatment of characters

When a character is read, parity check is performed if specified. When a parity error is detected the value 26 (decimal) is substituted. When parity checking is specified 7 least significant bits are taken as character value in the conversion.

The conversion is performed when conversion table address is nonzero as table lookup, giving a class and a value byte for each character.

```
Convtable/2 :     | class | value |  ; value 0
                          :
                  | class | value |  ; value "char"
```

Class = 0 normal character, the value is delivered (and echoed)

Class∧128 <> 0 attention character, status attention is delivered. Input is terminated. The value is neither delivered nor echoed.

Class∧64 <> 0 termination character, input is terminated with this character. If class includes special echo, this is performed, otherwise treated as normal character.

Class∧32 <> 0 special echo, the word class concatenate value bits 3 to 15 is taken as a displacement in words from convtabel/2 to a subtable containing:

convtable/2:

I

Class ıı value (3:15)

subclass	value
echo 1	echo 2
echo 3	------
-------	------
128	

Subclass ∧ 128 <> 0	erase current input buffer
Subclass ∧ 64 <> 0	erase last character (if any) from input buffer.
Subclass ∧ 32 <> 0	the value is not delivered.

Value gives the value delivered, and echo 1, echo 2, ... is a string terminated by 128, which is echoed on the echo channel (if any).

OUTPUT

Operation (0:7) = line

The text to be output should be terminated by a stop character (value 128).

STATUS

1B1	data set not ready
1B2	calling indicator
1B3	carrier off
1B5	attention char received
1B6	channel reserved by another user
1B8	buffer overflow
1B10	parity
1B11	break received
1B14	timeout.

General Description.

Control- and Output messages are accepted. Input messages are treated as Control messages.

Control.

Reservation and Conversion is accepted. Conversion table address should be a byte address. The characters to be output are converted as

conv. char: = byte (convtable + char);

Output Operation.

One mode of operation exist:

3: The converted characters are output to the incremental plotter.

The maximum size of the output blocks is 64 bytes. Only the last 4 bits in a byte are significant.
The meaning of the last four bits in the byte is:

decimal	binary	
0	0000	= step (+dy, 0)
1	0001	= step (+dy, +dx)
2	0010	= step (0, +dx)
3	0011	= step (-dy, +dx)
4	0100	= step (-dy, 0)
5	0101	= step (-dy, -dx)
6	0110	= step (0, -dx)
7	0111	= step (+dy, -dx)
8	1000	= step all pens up
9	1001	= pen 1 down, pen 2 and pen 3 up
10	1010	= pen 2 down, pen 1 and pen 3 up
11	1011	= not used

decimal	binary	
12	1100	= pen 3 down, pen 1 and pen 2 up
13	1101	= not used
14	1110	= not used
15	1111	= not used

If neccessary decimal 16 can be added to all data.

Status.

1B8	:	block error
1B14	:	timer

Execution Times

Note all timings for NOVA 1200

Interrupt		
dummy	92 µ	(std clear)
driver waiting	153 µ	(std clear)
Timer		
(no processes started)	149 µ	
+ Each process started	92 µ	
Wait		
int. pending	210 µ	
buffer pending	232 µ	
no activation	163 µ	
Wait interrupt		
int. pending	169 µ	
no actication	155 µ	
Sendmessage		
event waited	375 µ	
no activation	269 µ	
Wait Answer		
answer pending	318 µ	
answer not present	104 µ	
Wait Event		
event present	118 µ	
event not present	114 µ	
Send Answer		
answer waited	299 µ	
not activation	171 µ	

Send Answer

answer waited	299 µ	
not activation	171 µ	
send message + waitanswer		387 - 489 µ
send answer + wait answer		403 - 488 µ
total message traffic		790 - 977 µ
waitinterrupt		261 - 308 µ

next.operation (-waitevent)	40 µ
return answer (-sendanswer)	30 µ
clear	10 µ
setinterrupt	25 µ
setreservation	16 µ
setconversion	10 µ
conbyte (no conversion)	5,25 µ
(conversion)	21,00 µ
getbyte	14,40 µ
putbyte	27,45 µ
multiply	126 µ
divide	135 µ
(move with getbyte, putbyte)	64,85 µ / bytes

move (min) 32,40 + 10,50 * (bytes + 1) //2

(max) 121,05 + 23,70 * (bytes + 1) //2

(average) 76,8 + 17,10 * (bytes + 1) //2

bindec	478 µ
decbin	181 µ

getrec	(-inblock)	68 μ
	(+1.variable length field	+38 μ)
putrec	(-outblock)	96 μ
	(+ 1 variable length field	+315 μ)
wait transfer	(-waitanswer)	85 μ
	+ control	11 μ
	+ position check	23 μ
repeatshare	(-sendmess, waitanswer)	105 μ
	+ position (-sendmess,waitanswer)	75 μ
transfer		33 μ
inblock	(-transfer, waittransfer)	23 μ
outblock	(-transfer, waittransfer)	23 μ
inchar	(-inblock)	39 μ
outchar	(-outblock)	38 μ (50 μ)
backspace		15 μ
outend	(-outblock)	47 μ
outtext		53 μ * # chars
outoctal		332 μ

waitzone	(-outblock, -waittransfer) (2 shares)	100 μ
setposition	(-waitzone, -transfer)	49 μ
close	(2 * waitzone, -transfer)	150 μ
open	(-transfer)	47 μ