ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

C.N.R. - PISA

VIRTUAL INPUT/OUTPUT IN A VIRTUAL ENVIRONMENT

P.ANCILOTTI* R.CAVINA**

N.LIJTMAER*

Nota Interna n°1 - Serie Speciale

Convenzione CNR-ENI

Luglio 1971

*Istituto di Elaborazione della Informazione,
C.N.R. - Via S.Maria 46 - 56100 PISA, ITALY

**ENI-PRICE, Via S.Maria 83 - 56100 PISA, ITALY

Abstract

The purpose of this paper is to investigate the possibility of allowing an I/O channel, regarded as a special purpose processor, to execute a program allocated in the user's *virtual* space.

The environment is a multiptogramming-multiaccess system with a virtual memory implemented by paging.

For easiness of comparison, a standard system is sketched first, where the supervisor takes care of translating the channel program from virtual to real space.

Next, the proposed system is outlined, and the extra logical requirements for the channel are pointed out. The most relevant advantages are: I) The process information is treated in the virtual space, homogeneously; II) channel programs and their priorities can be dynamically modified; III) lower system overhead and greater flexibility are achieved.

tages connected with allowing access to DATM also to channels. We will make reference to this approach as *Virtual I/O in a virtual environment* [6]. We believe that, while these advantages might be not essential in a computer center context they *are likely to become relevant in real time or control process applications.*

The discussion carried on in the paper about real versus virtual I/O can be resumed as follows. If the channel is assumed to have direct access to the main storage and thus it does not share tha DATM, some difficulties are likely to arise:

- Since automatic storage allocation for programs is available, then the translation is guaranteed during execution from the address space to the memory space. This operational mode is not possible for the channel program.
- Since the channel program runs in the memory space and the user writes the channel program in the address space, the burden of translation and relocation is left to the user.
- The dynamic linkage between the program and the translated channel program, must also be handled by the user.
- The same situation arises for interrupt service.

The complexity of program management as outlined above has compelled the system designers to resolve these difficulties through the operating system [7-8-9-10].
This solution allows the programmer *to write both the program and the channel program in the address space.* However, the system overhead is increased. Furthermore we will see that the operating system requires translation facilities in both senses, thus the channel program cannot be *dynamically modified.*

If we allow access to DATM also to data channels, we get:

- An homogeneous treatment of process information in the virtual space.

. Possibility of dynamic modification of channel programs.

. Lower system overhead.

. Better dynamical utilization of main storage.

We are aware that our proposal implies a greater hardware sophistication of the channel. In the last section we will try to draw a first balance.

2 *REAL I/O IN A VIRTUAL ENVIRONMENT*

In this section, we will try to show briefly a possible solution an operating system may adopt.

Characteristics of this solution are:

- The user may write its channel programs in the address space.
- The Operating System handles real I/O operations and I/O interrupts.
- The Operating System provides an interface between the memory space and the virtual space.

We may distinguish three phases (see Flow-chart - fig.II)

2.1 *I/O Program initialization*

Operations performed in the I/O initialization phase are:

- Obtaining in the memory space the specific real channel program.
- Allocating the real channel program and the working area for its execution.
- Creating an I/O request and queuing it on the appropriate channel. This queue, of course, will be handled by an Operating System special routine.

For clearness, the initialization phase will be followed step by step (see flowchart fig. II).

2.1.1 *Channel Program generation and / or activation*

The facilities given to the user to perform I/O may change with the particular policies chosen by the Operating System.

For instance:

- The user may request an I/O operation supplying the parameters to one of the standard system macros. Afterwards, the macro-processor performs the subsitution by a subroutine. If a macro-generator is used, a functional transformation occurs.

  However, in a multiprogramming system, the loading techniques require dynamic initialization. Thus the program, the channel program, and the references lie in the address space.
- The user may request the activation of his own channel program written in his address space.

This two methods are not alternative, but they are generally implemented together in the same Operating System and may coexist in the same user's program.

2.1.2 *Translation*

The channel program, written in an intermediate language (L_i), lies in the virtual space and must be translated in an object language (L_o). This translation can be of different complexity. The translator must consider the following cases:

- An I/O command which involves a transfer of data to or from contiguous areas of virtual memory requires a sequence of data commands, one for each physical page, in order to accomplish the movement of data. Otherwise, the supervisor may reserve a non-paged buffer and perform an extra data transfer to / from the involved logical pages.
- Since there is no one-to-one correspondence between commands of the channel program in the intermediate language L_i and commands of the channel program in the object language L_o, the translator has to implement a table containing both logical and physical command

addresses.

In a multiprogramming context, each channel program may belong to different address space. The address space must then be recorded by the Supervisor. If an interrupt occurs during a command execution, the above information allows the Operating System to retrieve the logical address of the command.

. Every page required by the channel program for data or control information transfer must be locked in physical memory during the execution of the channel program for allowing the channel to work in cycle stealing mode. Then the O.S. must satisfy this condition.

2.1.3 *Creation of a request for a I/O task in the respective channel queue*

2.2 *Execution*

When the channel is not busy, the Operating System manages the queue and then activates the channel giving the address of the first command the channel must execute.

From now on, the channel and the CPU will be able to work in parallel.

2.3 *Channel interruption handling*

When the I/O operation is completed and an interruption occurs, the Operating System receives the control and then provides to the following tasks:

. All pages involved in data or control information transfer have to be unlocked and return to free storage In the case, the Supervisor must return the buffer to free state . Of course, transfer of input data from the buffer to the address space must be completed first.

- Status information is stored in the user's address space.
- The I/O task is unchained from the channel queue and the channel state is switched to "not busy".

3 *VIRTUAL I/O IN A VIRTUAL ENVIRONMENT*

In this section we outline the logical structure a system may have which implements a virtual I/O in a virtual environment. As already pointed out in the introduction, the basis of this method consists in allowing the channel to use a DATM. As a consequence, channel address references are in the virtual space. In our proposed implementation, the channel must have, in addition, the following technical characteristics for over all efficiency:

- The channel must contain information specifying to *which address space the channel program belongs.*
- From the set of all commands the channel singles out a subset of *privileged commands* controlling the data transfer.
- Two extra interrupt signals must be added to the standard ones, namely "*page fault*" and "*attempt of executing a privileged command*".

Also in this case, we can distinguish three phases (See Flowchart in Fig. III).

3.1 *I/O program initialization*

Operations that are performed by this phase are:

- Obtaining the channel program in the virtual space.
- Creating an I/O request and queue it on the appropriate channel.

More precisely, we have the following steps.

3.1.1 *Channel Program generation and/or activation*

As in the Real I/O in a virtual environment, the facilities available depend upon the Operating System [See paragraph 2.1.1]. However, it must be emphasized that the obtained channel program is now in the virtual space and its execution takes place without any intermediate processing.

3.1.2 *Creation of a block request for an I/O task in the respective channel queue*

The logical information to be stored in the block is:

- Indicator of the Address space which the channel program belongs to.
- Logical address of the next command.
- Channel Status and Channel program Status.

The Supervisor must take care of the management of the queue and of the effective activation of the Channel program.

3.2 *Execution*

Before beginning to describe a flowchart concerning the execution of a channel program, it is suitable to add some remarks:

- In what follows, we assume that *the channel program can be in three possible states:* WAIT, RUN and READY. In fact, when a channel program needs information which is not in main storage, then an interrupt from the channel to CPU occurs. When the CPU accepts this signal, the channel program is switched from "run state" to "wait state". The information associated with the channel is then stored in the queue, the channel changes its state to "not busy" and the Supervisor may assign the channel to another activity. Next, when the request of the channel program is satisfied, the Supervisor changes its state to "ready".

From now on, the channel program competes again for the channel. When activation is performed the channel program is switched to run state, and it restarts in the point it was interrupted.

- The channel program *is not necessarily resident in the main storage* during its execution. However, if the channel program has a high priority, the Supervisor may lock it in memory to minimize"page fault"interrupts and to avoid breaks. This degree of freedom contributes to the O.S. flexibility because the alternative between storage optimization and response time optimization can be solved case by case by the O.S. scheduler.
- During execution of a *privileged command* the involved physical pages must be locked and their addresses allocated in the associative memory of the DATM. Afterwards, the data transfer takes place in cycle stealing mode.

  If the command is not privileged the possible access to main storage must also be done through the DATM, but not necessary through its associative memory. It may be done for instance by a table searching.
- Since in a multiprogrammed system the CPU and the channel may work in overlap on differents programs the channel must have information relative to the address space the channel program belongs to.
- The absence of an intermediate translation phase of the channel program allows it to be dynamically modified.

Let us now consider briefly what takes place during execution of a channel program (See flowchart fig. IV). The channel knows the logical address of the next command. This address (i) may or (ii) may not be in physical memory.

(I) In this case the operation sequence of the channel

is :

- The channel fetches the command and analizes the operation code.
- If the operation code does not belong to a privileged command, the channel analizes if it makes reference to an operand address. If it does not or if the logical address referenced is presently in physical memory, the channel executes the command. Otherwise, if the referenced logical address is not in physical memory, th n a "page fault" interrupt occurs and the CPU proceeds like in case (II).
- The operation code is a privileged one. Note that we allow a privileged command to control the data transfer from (to) *more than a single logical page*.
  When a command of this type is recognized, a "privileged command" interruption occurs. Then the Supervisor takes care of the fetching of the logical pages and of their placement in physical memory. Furthermore, the involved physical pages are locked by the Supervisor, and their addresses are allocated in the associative memory of the DATM.
  This method allows the channel to work in cycle stealing mode. Moreover, the physical pages, which have been dynamically assigned, need to be locked *only during the data transfer process*. The same property holds for their addresses in the associative memory.

(II) If the address is *not* in physical memory a "page fault" interrupt occurs from the channel to the CPU and the relative O.S. routine is activated. This routine provides:

- A request for a transfer of the referenced logical pages into main storage.
- To store in the block queue all the information related to the channel in order to guarantee the restart of the channel program in the same point where it was

interrupted.

- To switch the channel program to wait state. Therefore, the channel becomes "not busy" and then a "ready" channel program of the queue can be activated by the supervisor.

From now on the channel and the CPU may run in parallel.

Two remarks are pertinent:

- We need to lock the pages *only during execution of a privileged command.* Therefore, if a channel program requires the execution of more than one privileged command, the memory utilization is increased.
- The Supervisor *unlockes the pages* when the first interrupt takes place, even if the channel program execution is not completed.

3.3 *Channel and interruption handling*

When the channel program is completed, the Supervisor provides to release the System resources related to this task.

4 *CONCLUSION*

Both systems analized, Real and Virtual I/O, allow the user to write channel programs in his virtual space. However, in Virtual I/O in a Virtual environment, the channel program does not undergo any intermediate transformation, it may be dynamically modified and it must not be resident during execution.

Since in Virtual I/O the pages are not locked during execution of the whole channel program, but only during the effective data transfer,the dynamic utilization of main storage is improved.

The total system overhead decreases. In Real I/O there is a translation routine in the initialization phase (paragraph 2.1.2) and an interrupt handling routine to reflect in the address space what takes place in the real space (paragraph 2.1.2). In virtual I/O these routines are not required. The system overhead related to fetch, placement and replacement policies of the involved I/O pages, is not eliminated but is distributed during the channel program execution.

Since the channel program may give an interrupt, the channel may be assigned to another task of its queue. Afterwards, the channel program restarts in the same point it was interrupted. Therefore, the priority of the process assigned to the channel may change dynamically and there is no idle time for the channel.

We point out that the advantages seen above are obtained through an increase in the sophistication of the channel logical capabilities. Thus by more complex hardware structure. Hardware complexity is also increased by the fact that DATM is

shared by more processors, while its associative memory may or may not be assumed as pooled by the processors.

This paper is taken as a starting point for a future development. We are aware that an actual implementation or an exhaustive simulation could make apparent other nonobvious properties of the system and could allow a detailed design and cost analysis.

APPENDIX A

The CP - 67 (Control Program 67) runs on a computer IBM 360 - Mod. 67 and provides the time-sharing part of the System. The environment of CP-67 consists of "virtual machines". A virtual machine is a functional simulation of a real computer and its associated I/O devices.CP-67 builds and mantains for each user a virtual System /360 machine of a predescribed configuration. The user and his programs cannot distinguish the virtual /360 from a real System /360, but in fact his virtual machine is really one of many that CP-67 is managing. CP allocates the resources of the real machine to each virtual machine, for a "slice" of time and then moves on to the next virtual machine to guarantee time-sharing.

One of the resources CP-67 must handle is I/O channels. Three stages are distinguished in CP-67 to process a user channel I/O request:

i) *Analysis of the request performed by a virtual machine*

The virtual machine runs in problem state. Only CP-67 runs in supervisor state . When the execution of an I/O instruction of the virtual machine is attempted, an I/O interrupt occurs. This is the way to denote to CP-67 an user I/O request. Then, CP-67 provides to determine the type of the I/O instruction and translates the virtual addresses of the channel of the control unit and of the devices into the real addresses.

ii) *Translation of the channel program associated to the request from the virtual space to the real one*

If the involved instruction is a SIO (START I/O) the VCAW (Virtual Channel Address Word) point to the first command of the virtual channel program. Then CP-67 must translated the virtual channel program to the specific real one. This task is performed in three phases:

- The *Scan phase* analizes the virtual channel program to determine the total storage requirement of the real channel program. Pages involved in this operation are locked in main storage.
- The *Translation phase*, re-examines the virtual channel program and translates it into a real one. Some TIC (Transfer In Channel) commands that cannot be immediately translated are flagged for later processing. Read or Write commands that spacify data crossing page bounderies are translated into several commands each for one page only.
- The *TIC-SCAN phase* scans real channel program for flagged TIC commands and creates new virtual commands accordingly.

Scan phases processing is then restarted.
When these three phases are completed, the virtual Channel Address Word is replaced by the real one. An I/O TASK Block is created and queued for execution on the specific real channel queue.

iii) *Execution of the channel program in the real space and I/O interrupts reflection to the virtual machine*

When a "Channel End" Interrupt occurs the real channel status word is translated into the virtual one; the involved pages are unlocked. The I/O task Block is unchained from the real channel queue and retur-

ned to free storage.

A deep treatment of I/O in CP-67 is founded in the specific Bibliography [7].

APPENDIX B

MULTICS INPUT/OUPUT SYSTEMS

MULTICS (Multiplexed Information and Computing Service) is a comprehensive, general purpose programming system which is being developed as a research project. The Multics was implemented on GE 645 computer. This system is a multiprogramming-multiaccess system with a virtual memory for each user. It involves two-dimensional addressing with segmentation and paging. Conversely, the I/O Channels deal with physical addresses.

The MULTICS I/O System has been designed to regard the I/O resource needed to complete any given I/O operation not as a real or physical resource, but as a *virtual* I/O resource. This virtual resource is described in terms of the functions it must be capable of performing and it is mapped by the system, *at run time*, into a particular real resource, using whatever I/O device is available and convenient. However, the user may decide what I/O devices he wants used. In the first case the user codes an I/O operation independently from the type of device:

Then, the Operating System takes care of:

. Interpreting the user's I/O request and determining the type of the device associated with the specific I/O request.

. Compiling the channel program.

. Monitoring the channel execution.

Interrupts and recognition of completion task are then handled.

Two types of channel are attached to the GE 645 namely: the Indirect Adapter Channel and the Direct Adapter Channel Speed, number of registers, and logical capabilities are different.

Inizialization of channels, execution of channels programs and interpretation of their status information is achieved, in the GE 645 system, through a peripheral processor, called GIOC. (Generalized I/O Controller, see fig. V).

The access from the channels to the main memory takes place directly.

Therefore, the *DATM is not shared*, neither by the GIOC nor by the channels and we say there is a *real I/O in a virtual environment.*

The user has at his disposal macros like WRITE, READ, ATTACH etc. By means of this macros the user requires the I/O Supervisor Service.

Three modules may be distinguished in the I/O Supervisor System:

i) I/O Switch
ii) Device Interface modules (DIM)
iii) GIOC Interface module (GIM)

The I/O Switch calls the target DIM. Then, it determines the type of peripheral device required. There is one DIM for each specific type of peripheral device. The DIM compiles a channel program in the virtual space. It takes care of the technical characteristics of the particular device, and thus converts a device independent request into a device dependent one.

The DIM calls the GIM for allowing it to perform its various communication tasks. The GIM is a software module controlling the GIOC. The GIM's work areas and I/O buffers must be wired down because the GIOC can *only reference real addresses*. The GIM takes care of :

. Selecting the specific real device and its associated

channel.

- Translating the Channel program from the virtual space to the real space and allocating it in the GIM working area.
- Executing the channel program.
- Copying the channel status in the DIM's virtual space.
- Handling all I/O interrupts.

Finally the information flow through the modules of the MULTICS I/O System, I/O Switch→DIM→GIM→GIOC is illustrated in figure VI.

BIBLIOGRAPHY

[1] Denning, P.J. — *Resource allocation in multiprocess computer systems*, MIT - Project MAC (Ph. D. thesis) MAC-TR-50.

[2] Arden, B.W., Galler, B.A. O'Brien, T.C. Westervelt, F.H. — Program and addressing structure in a time-sharing environment. *Journal ACM 13, 1* (Jan, 1966); 1-16.

[3] Denning, P.J. — Virtual Memory *Computing Surveys-ACM*, Vol. 2, n°3, Sept. 1970.

[4] Dennis, J.B. — Segmentation and the design of multi-programmed computer systems. *J.ACM 12, 4* (Oct. 1965), 589-603.

[5] Arden, B.W. — Time Sharing Systems : A Review - The University of Michingan - Engineering summer conferences. *Computer and Program Organization Advanced Topics* - June 19-30, 1967.

[6] Smith, A.A. — "*Input/Output in Time-Shared, Segmented, Multiprocessor Systems*" Project MAC Report, MAC-TR 28, Master thesis.

[7] *CP/PROGRAM Logic Manual* — IBM Y20-0590 Cambridge Scientific Center.

[8] Organick, E.I., — *A Guide to MULTICS for subsystem writers - Chapter VIII - The input - output System* (DRAFT) - Nov. 10-1970. Project MAC.

[9] Ossanna, J.F., Mikus, L.E. Dunten, S.D. — "Communications and input/output switching in a multiplex computing system"*Proceedings Fall Joint Computer*

Conference, 1965, pag. 231-241.

[10] Vermillion, W.H. Channel command structure to operate in a paged environment - *IBM Technical Disclosure Bulletin*, Vol. 13 - n°5 Oct. 1970.

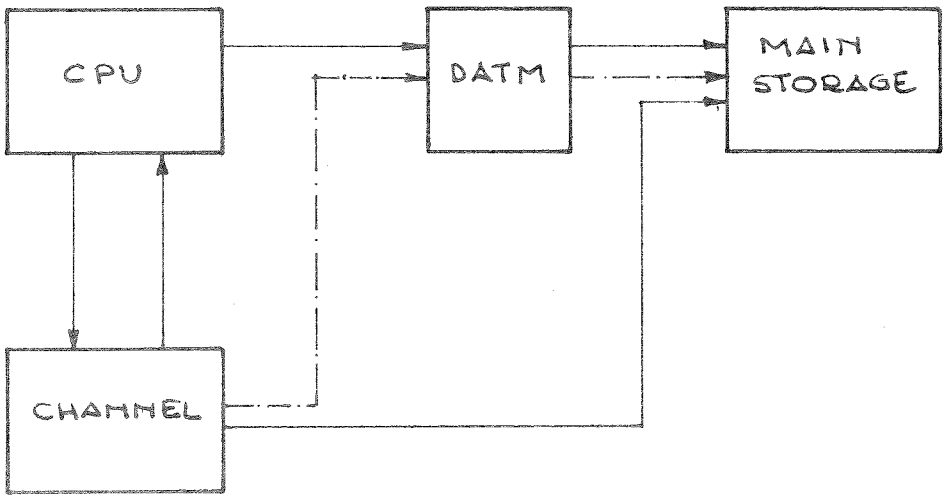

FIG. 1

REAL I/O IN A VIRTUAL ENVIRONMENT

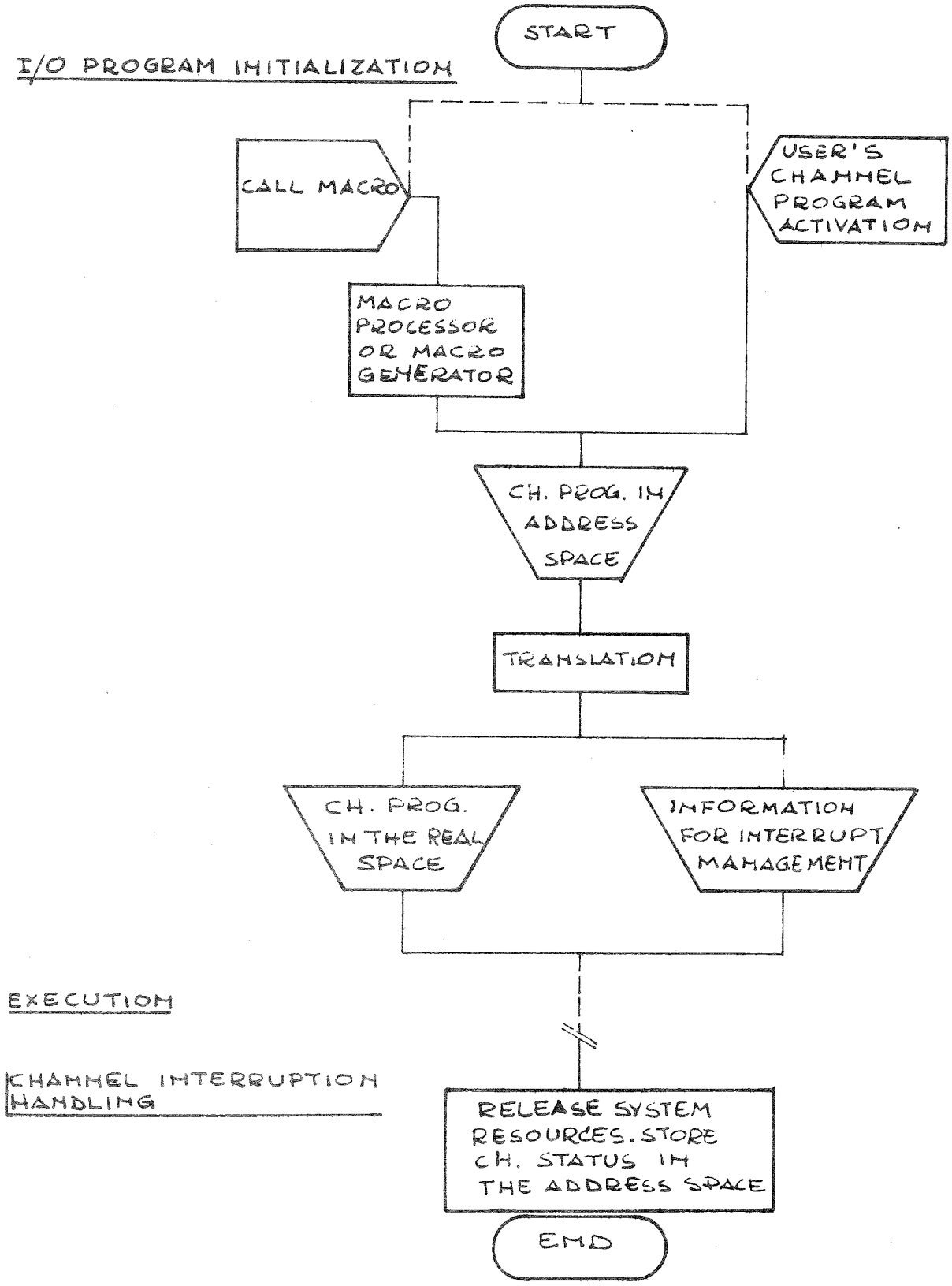

FIG. II

VIRTUAL I/O IN A VIRTUAL ENVIRONMENT

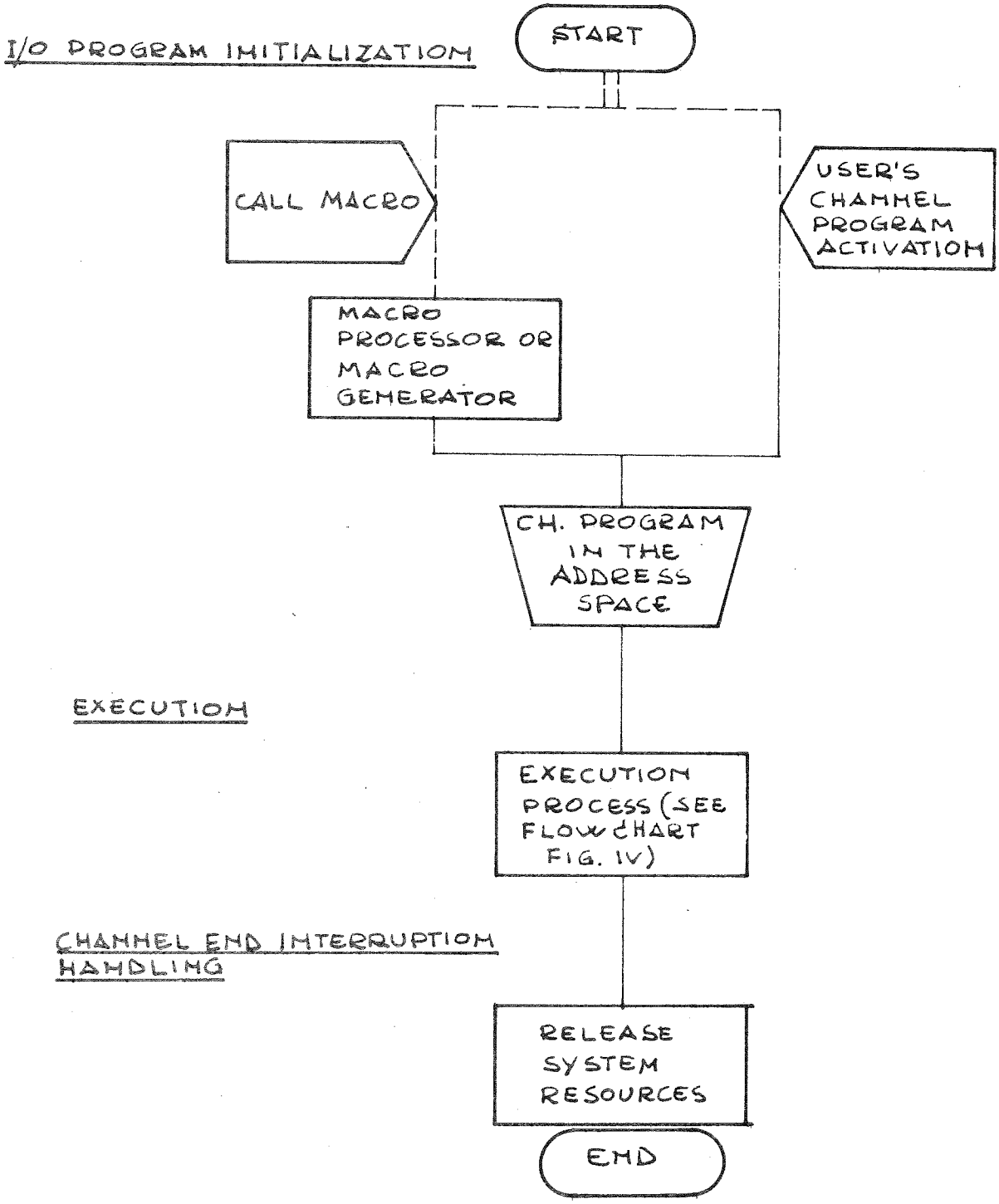

FIG. III

EXECUTION

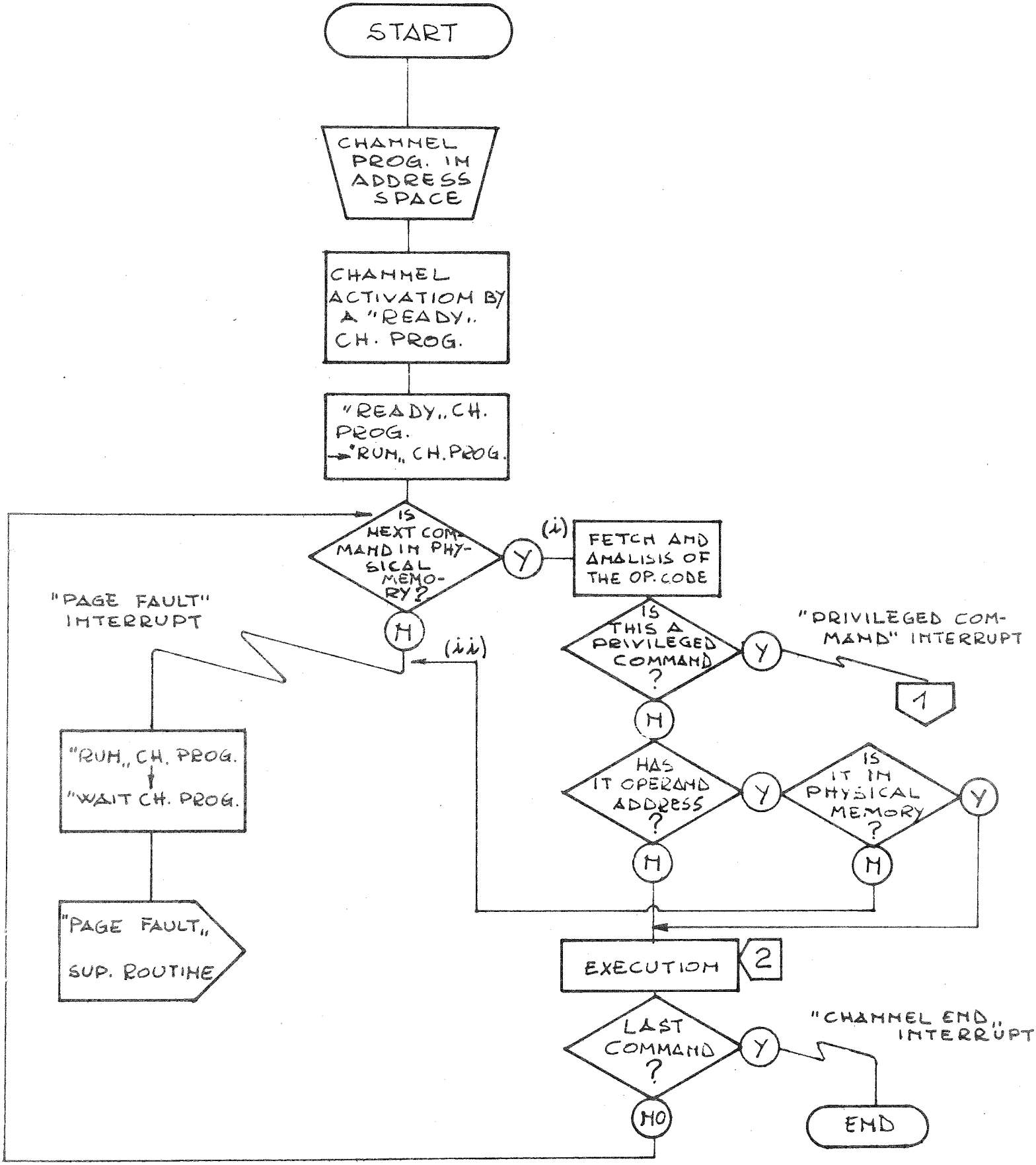

FIG. IV2

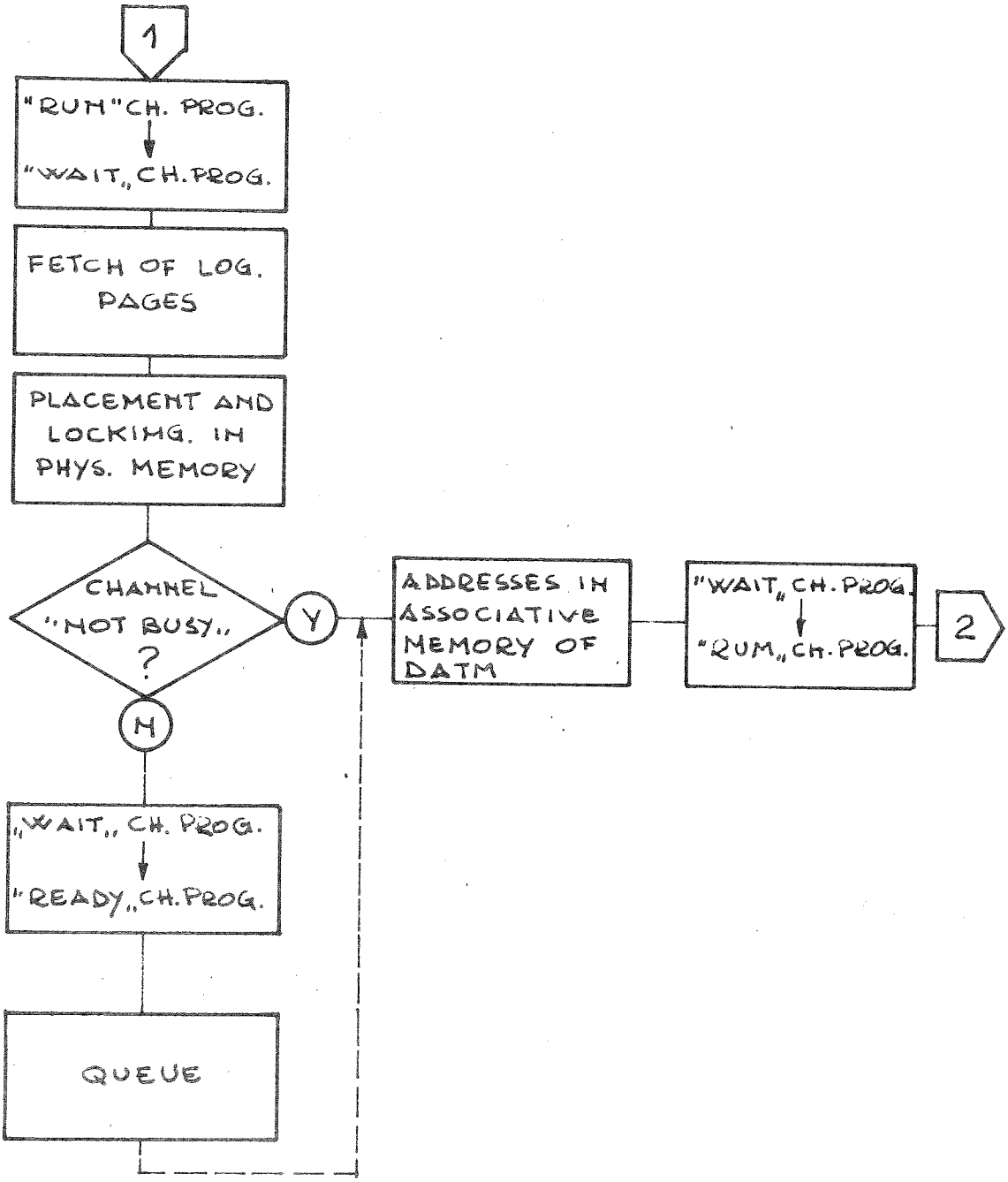
1
"RUN" CH. PROG.
"WAIT" CH. PROG.
FETCH OF LOG. PAGES
PLACEMENT AND LOCKIMG. IM PHYS. MEMORY
CHANNEL "NOT BUSY" ?
Y
N
ADDRESSES IM ASSOCIATIVE MEMORY OF DATM
"WAIT" CH. PROG.
"RUN" CH. PROG.
2
"WAIT" CH. PROG.
"READY" CH. PROG.
QUEUE

FIG. IV b

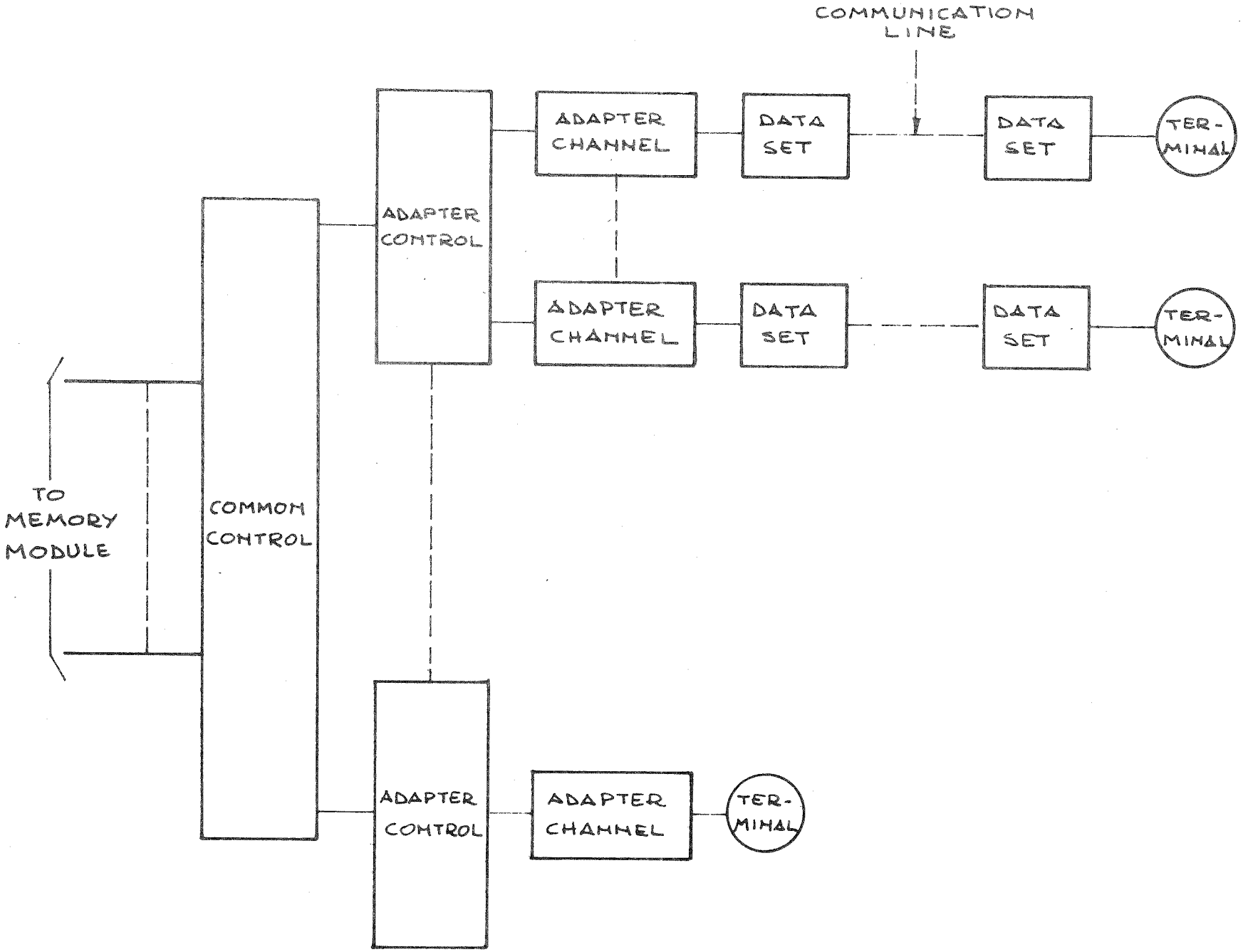

FIG. V - FUNCTIONAL ORGANIZATION OF THE INPUT/OUTPUT CONTROLLER -

I/O PROGRAM INITIALIZATION

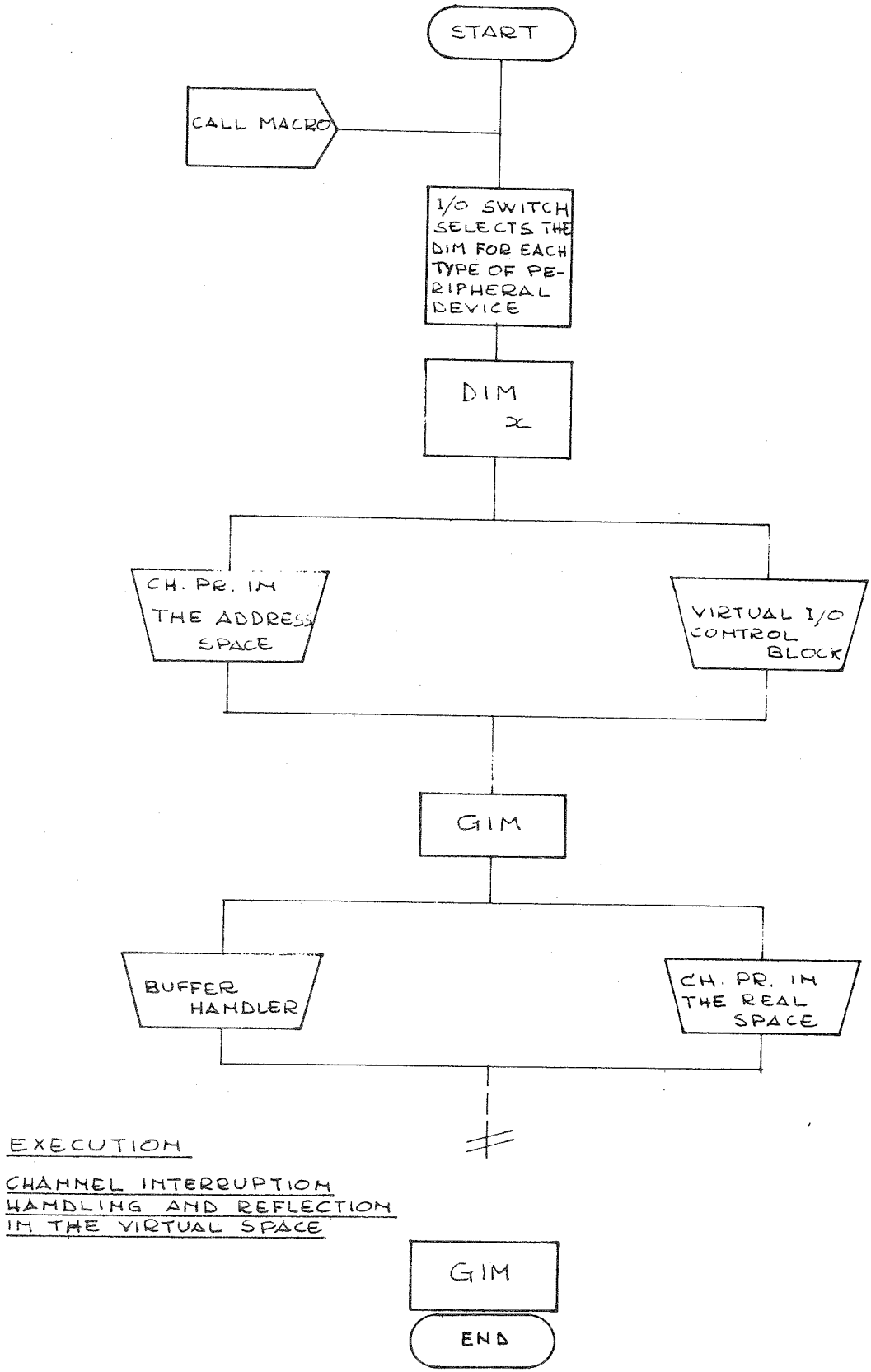

FIG. VI