KFKI-1978-17

Molnár Gábor

L. BÜRGER
Z. CSÖRNYEI
Á. MÁTYUS
J. PÉTER
G. SZABÓ
É. VÉGH
E. ZOBOR

PROCESS-24K
AN EFFICIENT PROCESS CONTROL SYSTEM

Hungarian Academy of Sciences

CENTRAL
RESEARCH
INSTITUTE FOR
PHYSICS

BUDAPEST

KFKI-1978-17

PROCESS-24K
AN EFFICIENT PROCESS CONTROL SYSTEM

by

L. Börger, Z. Csörnyei*, Á. Mátyus*, J. Péter,
G. Szabó, E. Végh, E. Zobor

Central Research Institute for Physics
H-1525 Budapest P.O.B. 49 Hungary

* Computer and Automation Institute, Budapest

HU ISSN 0368 5330
ISBN 963 371 386 2

Abstract

PROCESS-24K is an efficient real-time control system - for the R-10, R-12 and MITRA-15 computers. This system provides

- data acquisition to about 2000 analogue or digital variables,
- high level communication for up to 4 technological operators,
- loggings of different types,
- alarm analysis,
- adaptive control of a process.

The usual measurement and control problems listed above are solved by table controlled tasks of the system and the user has only to specify their operation by filling out these tables. This is done by means of the PROCESS high-level process control language and its compiler generates the appropriate tables. The structure of the control tables, the operation of the different programs, and the performance of the system will be treated in detail; moreover the problems of the system generation are also discussed. The most important features of PROCESS-24K are summarized in the Appendices.

Résumé

Notre reportage décrit le PROCESS-24K, système temps réel de contrôle des processus industriels à grande puissance, pour les ordinateurs R-10, R-12 et MITRA-15. Ce système assure la possibilité de

- traiter environ 2000 variables analogiques ou numeriques,
- informer max. 4 opérateurs technologiques à haut niveau,
- remplir différents journaux,
- analyser des alarmes, et
- contrôler un processus industriel d'une manière adaptive.

Ces problèmes usuels dans la technique de mesure et de contrôle sont resoulus par les tâches du système, dont l'operation l'utilisateur ne doit spécifier qu'en remplant des tablaux de contrôle à l'aide d'un langage à haut niveau pour le contrôle de processus industriels, PROCESS. Les structures des tableaux de contrôle, les operations des programmes différents et la puissance du système seront traitées en détail, ensuit les questions de la génération du système seront aussi discutées. Les caractèristiques les plus importantes du PROCESS-24K sont assemblées à l'Appendice.

Összefoglalás

Riportunk ismerteti a PROCESS-24K hatékony folyamatirányitó rendszert, amely az R-10, R-12 és MITRA-15 számitógéphez használható. Ez a rendszer lehetőséget biztosit

- kb. 2000 analóg vagy digitális változó kezelésére,
- max. 4 technológus operátor számára magasszintü kommunikációra,
- különböző naplózások ellátására,
- alarm analizisre és
- egy folyamat adaptiv vezérlésére.

A felsorolt szokásos mérési és irányitási feladatokat a rendszer táblázat vezérelt taszkjai oldják meg és a felhasználónak csupán ezek müködését kell specifikálnia a vezérlő táblázatok kitöltésével. Ehhez a rendszer a PROCESS magasszintü folyamatirányitó nyelvet realizálja és ennek forditó programja generálja a megfelelő táblázatokat. Részletesen tárgyalni fogjuk a vezérlő táblázatok szerkezetét, az egyes programok müködését és a rendszer teljesitőképességét, továbbá a rendszer generálás kérdései is ismertetésre kerülnek. A PROCESS-24K rendszer legfontosabb sajátosságait a Függelék tartalmazza.

АННОТАЦИЯ

Сообщение описывает высокоэффективную систему управления технологическими процессами PROCESS-24K, которая может применяться на ЭВМ-ах типа R-10, R-12 и MITRA-15. Система обеспечивает возможность:

- охватывать примерно 2000 аналоговых и цифровых параметров;
- коммуникацию на высоком уровне максимально с 4-мя операторами технологического процесса;
- составления различных протоколов и дневников;
- анализа алармов;

Указанные выше функции сбора данных и управления процессом выполняются различными таблицами. Для определения задач системы достаточно заполнить эти таблицы. Чтобы облегчить эту задачу система оснащена символическим языком PROCESS, ориентированным для решения задач сбора данных и управления процессом. В сообщении детально описываются состав и характер управляющих субпрограммами таблиц, функции отдельных субпрограмм, мощность системы управления PROCESS-24K и вопросы генерации ее на ЭВМ. Важнейшие характеристики системы даются в Приложении.

Acknowledgement

The development of the PROCESS-24K was based on the PROCESS-8K system. Throughout the development the authors of this report have got a considerable help from the coworkers of the Computer and Automation Institute of the Hungarian Academy of Sciences where the PROCESS-8K system was constructed.

It is a pleasure for the authors to express their gratitude for this help and to thank Mr. S.Keresztély, the chief designer of the PROCESS-8K system, for his valuable advice and continuous encouragement during the development work.

CONTENTS

CHAPTER 1.

INTRODUCTION

In the early 70s the utilization of digital computers for industrial process control increased dramatically and this in turn initiated a great development in industrial real-time programming. Although in the early stages the assembly coding predominated the industrial application area, now various types of real-time languages are used at almost every installation. The main reasons for this lie in the growing size and complexity of the problems and in the drastic decrease in the price of the computing hardware |1|.

Two tendencies in the use of high level real-time languages in the industrial environment can be observed. One of these trends is in favour of existing widely accepted high level languages, and provides real-time extensions to them. The others prefer new problem oriented languages which allow the user of an industrial computer to develop his system easily without having a detailed knowledge of the used computing means.

Universal high level languages, which are completed with real-time extensions, are FORTRAN |2| and BASIC |3|. In these languages the extensions are in the form of subroutines called by a CALL statement. The extensions are needed to solve the following problems

- starting of a program with a given delay,
- starting of a program at a given time,
- waiting for a given interval,
- initiating/terminating a program when a given condition is fulfilled,
- handling of a real-time peripherals /A/D converters, digital input/output, etc./
- handling of different types of files.

The most serious problems arise in the input/output organization where there is often a strong interaction between the machine independent language and the configuration dependent operating systems, so the I/O operations are not always compatible on different machines.

A considerable number of problem oriented languages have been developed recently, e.g.: INDAC |4|, PROCOL |5|, LTR |6|, PEARL |7|, CORAL |8|, etc. It is generally required that these languages

- should need only a short learning time from the process engineers,
- should not need the user to have any knowledge of the internal structure of the computer, of number representation, of timing, etc.,

- should provide efficient restart procedures and diagnostic aids,
- should give the possibility of on-line modification.

With such languages the program writing time and errors are reduced since the real-time problems are solved by the language. The debugging and the program modification are quite simple and well documented. The language efficiency is considerably good, about 1,3-1,5 compared to assembly programming.

More simplified types of problem oriented languages are the so called format defined languages e.g. BICEPS |9|. In these systems the user has only to fill in standardized forms. The main advantage of these languages is in their simplicity. The standardized form may prevent the user from forgetting some important points in the description of this problem. On the other hand these programs are not directly transportable to another computer though the required effort for the transcription is generally not too great. Using a high level language, it is almost unavoidable that some parts of the application software be written in assembly language |11|.

There are three reasons for using assembly coding

- special machine instructions are sometimes needed to directly drive the hardware,
- for possible re-entrance /although CORAL 66 for example provides re-entrance as well/,
- when a program path is very frequently used, the time minimalization is critical.

For these reasons, the possibility of inserting assembly segments in a high level language is highly recommended. The experience of real-time programming in an industrial environment has proved that a problem is better solved by a process man, who knows the process well but has a very limit knowledge of programming, rather than by an experienced programmer, who may not understand the process. This fact has initiated the development of simple but effective operating systems, which are not general purpose systems but rather process oriented ones |10|. In general a process oriented operating system incorporates the compiler of a given high level language and tries to simplify the programming in every possible way.

Nuclear power generation is an industrial area where computers are used for a long run. There is no uniform opinion on the role of the computers in nuclear power plants at present. In Canada process computers have been used in plant control for several years |12|, whereas the situation in the USA is that no control functions are permitted and process computers are used only in data acquisition and operator information systems |13|. The practice in England lies somewhere between these two experiences. Process computers provide automatic data reduction and eliminate manual data logging, moreover, they interlock some plant operation when the proper conditions are not fulfilled |14, 15|. Instead of direct digital control, the sequence control of the operator's activity is preferred.

Until now in Hungary two Hungarian made small computers have been used in process control: the TPA-i which is program compatible with PDP-8 and R-10 - the licensed version of the French MITRA-15.For the R-10 computer two process control systems were originally developed: a process oriented operating system /PROCESS-8K |16, 17, 28, 29|/ incorporating the PROCESS problem oriented language, and a format defined language /PROCESS-16K |18|/ which operates in the Process Control Monitor operating system |19|.

In the Central Research Institute for Physics, Budapest, several industrial computer applications have been developed and installed successfully. The first installation of this type was the block monitor system completed in 1975 of the Danube Thermal Power Plant |20|. Since this time a number of other similar installations have been completed and at present there are others under construction. In the early 70s a research project was launched, the aim of which was of establish a closed-loop computer control system on the WWR-SM[x] research reactor of our Institute. This project was supported by the State Office for Technical Development and by the National Atomic Energy Commission.

The PROCESS-24K process oriented operating system, incorporating the PROCESS problem oriented language, has been designed in connection with this project. The configuration is based on a R-10 computer of the VIDEOTON Computer Factory. In the first step of the project /1976/ the PROCESS-8K system was used. This needs only 8K words core memory.

Although the performance of the PROCESS-8K would have been sufficient to meet the requirements of the project, the PROCESS-24K system has been developed in order to provide a system which is able to be used in Nuclear Power Plants and in similar fast and dangerous installation. Throughout the development the essential, very progressive characteristics of the PROCESS-8K system[xx] have been retained in order to maintain the upward compatibility with the smaller PROCESS-8K system

---

x

A tank type light water moderated research reactor with 5 MW thermal power, the primary and secondary coolant circuits are coupled via two heat exchangers, the energy is absorbed in a cooling tower.

xx

These progressive characteristics are: the incorporated PROCESS language, the on-line loading/modificating properties of the system, the very efficient monitor, logging subsystem, etc.

This means that every application program of the PROCESS-8K can be used without any modification in the PROCESS-24K system.

The most significant advantages of the PROCESS-24K compared to the PROCESS-8K are the following:

1/ The generated data base is core resident instead of disc resident. This feature results a great increase in the processing speed.

2/ The PROCESS language is completed by internal functions and it is possible to call external functions as well.

3/ Assembly coding is also permitted, and assembly modules can be linked to tasks written in the PROCESS language.

4/ Alarm analysis - providing a deeper insight into the actual process - is an inherent part of the system.

5/ Under normal operating conditions of a plant, the goal of the computer control system is some kind of optimization. In anomalious situations, however, the aim of the control is dictated by safety aspects, i.e. some high priority emergency algorithms have to enter and other optimization tasks must stop operating. PROCESS-24K provides a framework for realizing such a reconfiguration.

6/ Strong emphasis has been laid upon man-machine communication to help the operator in unexpected situations. This goal is achieved within the possibilities of alphanumeric displays on the screen of which the operator can call

- alarm lists,
- alarm trees,
- technological logs

or he can initiate a dialogue.

7/ PROCESS-24K provides automatically refreshed data presentation on lamps, numerical indicators or on any other type of digital display equipment.

The price of the advantages listed above is the 24 Kword core memory, but now - when the price of the memory is decreased considerably - it seems to be not too serious.

CHAPTER 2.

GENERAL DESCRIPTION

2.1. Hardware configuration

PROCESS-24K needs the following hardware

- R-10, R-12 or MITRA-15 central processor with 48 Kbyte operating memory and with floating point arithmetic unit
- fixed head disc with 800 Kbyte capacity /DISCMOM -EC-5060 or SAGEM FEX-3/
- real-time clock
- console typewriter
- two alphanumeric display units /VT-340/
- tape reader and punch
- real-time measuring system with

  1 - 4 integrating A/D converters /71921/
  1 - 64 32-line analogue multiplexers /71912/13/
  1 - 16 8x16 bit digital inputs /71950/51/
  1 - 16 8x16 bit digital outputs /71960/61/
  1 - 16 16x8 bit polarized relay output /71970/

The following units can be handled by PROCESS-24K, but they are optional

- 2 logging typewriters /Console 260/ or matrix printers /DZM-160/
- 1 line printer /VT-343/
- 3 magnetic tape units
- 2 additional alphanumeric display units.

The hardware configuration can be seen in *Fig. 1.* Since PROCESS-24K is a multiprogrammed system and the individual programs use different interrupt priority, the following levels are required. /see next page/

In Appendix 1 we summarize all constraints against the hardware /addresses, DVA words/. Each entity is selectable by a jumper in the corresponding hardware unit, so this constraint is not a very serious one.

IT level	Program name	Peripheral
0	BACKGROUND	-
1	COMLOG	-
2	OPER	74.880+
3	ALARM	-
4	NIXI	-
5	PULT	-
6	ASR	CONSOLE
7	MPX	DISPLAY
8	LPT	PRINTER
10	CLA	TYPEWRITER
11	ALDYS	74.880
13	PTP	TAPE PUNCH
14	PTR	TAPE READER
15	ANAL	-
16	MT	MAGN. TAPE
17	FELD	-
18	MEAS	74.880
19	RELE	-
21	HWIT	74.880
24	TIMER	CLOCK
26	DISK	DISC
30	PWUP	-
31	PWDOWN	-

\+ 74.880 is an interrupt collector card which gathers 16 individual interrupt request lines into one interrupt priority level.

2.2. System architecture

It is usual to divide process control software into two main categories, viz.

- system programs /operating system, high level languages and different programming aids/
- application programs /each of which is unique in every application/.

In our opinion this approach reflects the general attitude of the computer system's suppliers, so it corresponds to the boundary of the responsibility instead of to the logical structure of such a system. Vernel |21| suggests three categories:

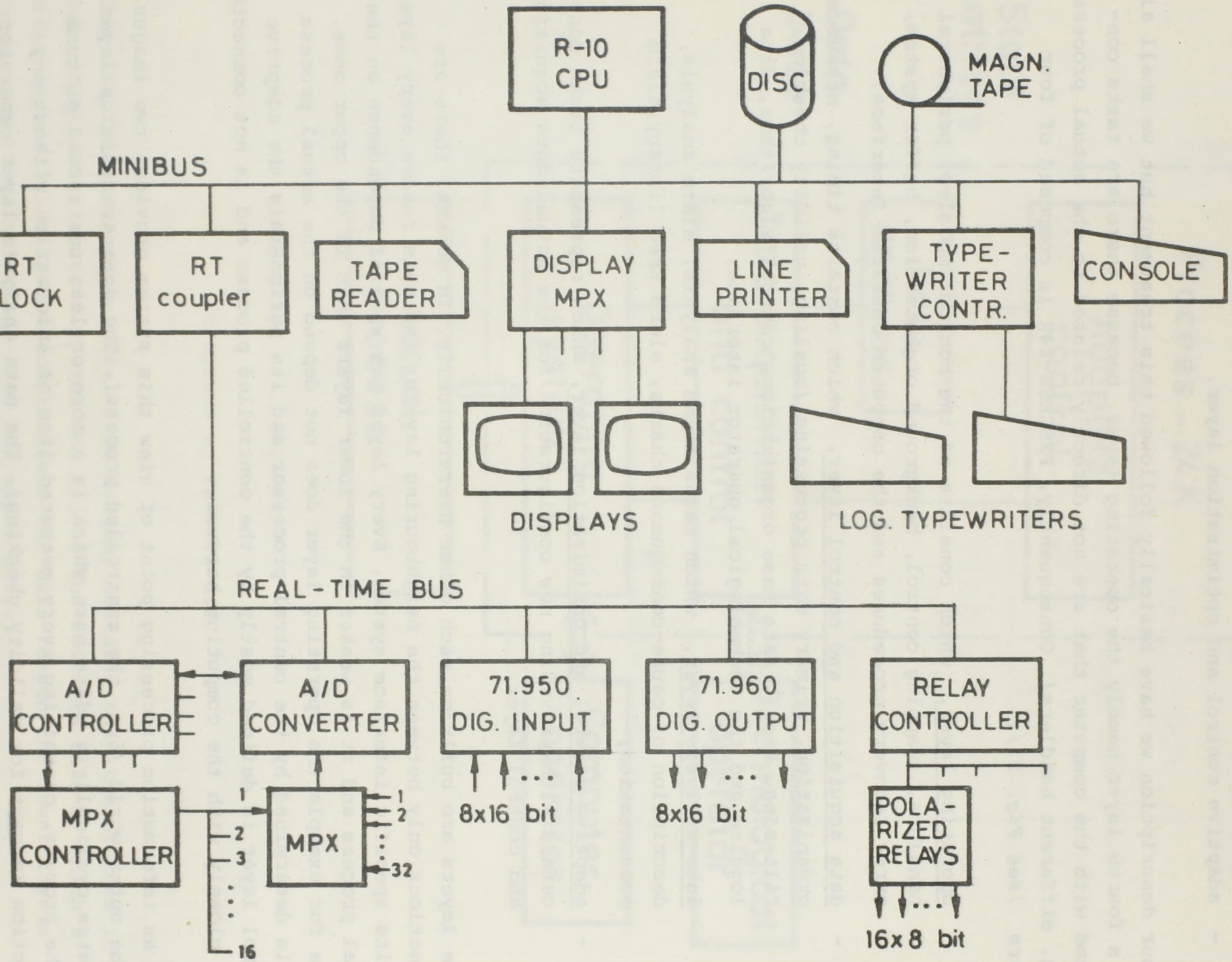


Fig. 1.

Hardware configuration for PROCESS-24K.

- data logging layer
- data analysis layer
- adáptive control and optimization layer.

In our description we have basically followed this treatment but we shall also use a fourth layer namely the operating layer, because there are tasks connected with the computer that are not directly related to the actual process /e.g. different handlers/. Consequently, PROCESS-24K is composed of four layers /see *Fig. 2.*/:

- operating layer, which consists of the monitor modules, peripheral handlers, swapping control, background organization, buffer system, error recovery procedures and the computer operator interface;
- data acquisition and control layer, which contains timing, measurement organization, primary data processing /scaling, validity checking, filtering, etc./, data base organization, control algorithms, data logging and the technological operator interface;
- data analysis layer, which means trend analysis, alarm analysis, description of cause-consequence charts, alarm tree library,alarm presentation;
- adaptive control and optimization layer, which represents tasks concerned with providing new configuration for the actual data acquisition and control layer.

These layers are built on each other hierarchically. In general there are connections only between the neighbouring layers; for this reason every layer has its specific interface system. Every layer has special dependence on the actual process and it is weaker in the lower layers than in the upper ones. While for example the operating layer does not depend on the actual process and is determined by the central processor and its peripherals the adaptive control layer is defined mostly by the controlled process and is not connected very closely with the computing hardware.

From an information processing point of view this system provides two images of the outer world /i.e. the controlled process/. The data acquisition layer up-dates cyclically a data base which is a more or less unstructed picture of the process, containing every measured item of information without any deduction /except for validity checking/. The data analysis layer generates a structured picture of the process so this image depends not only on the measured quantities but on the ordering principle too. Consequently this picture is more abstract and condensed than the former one; at this level the process is described by state matrices.

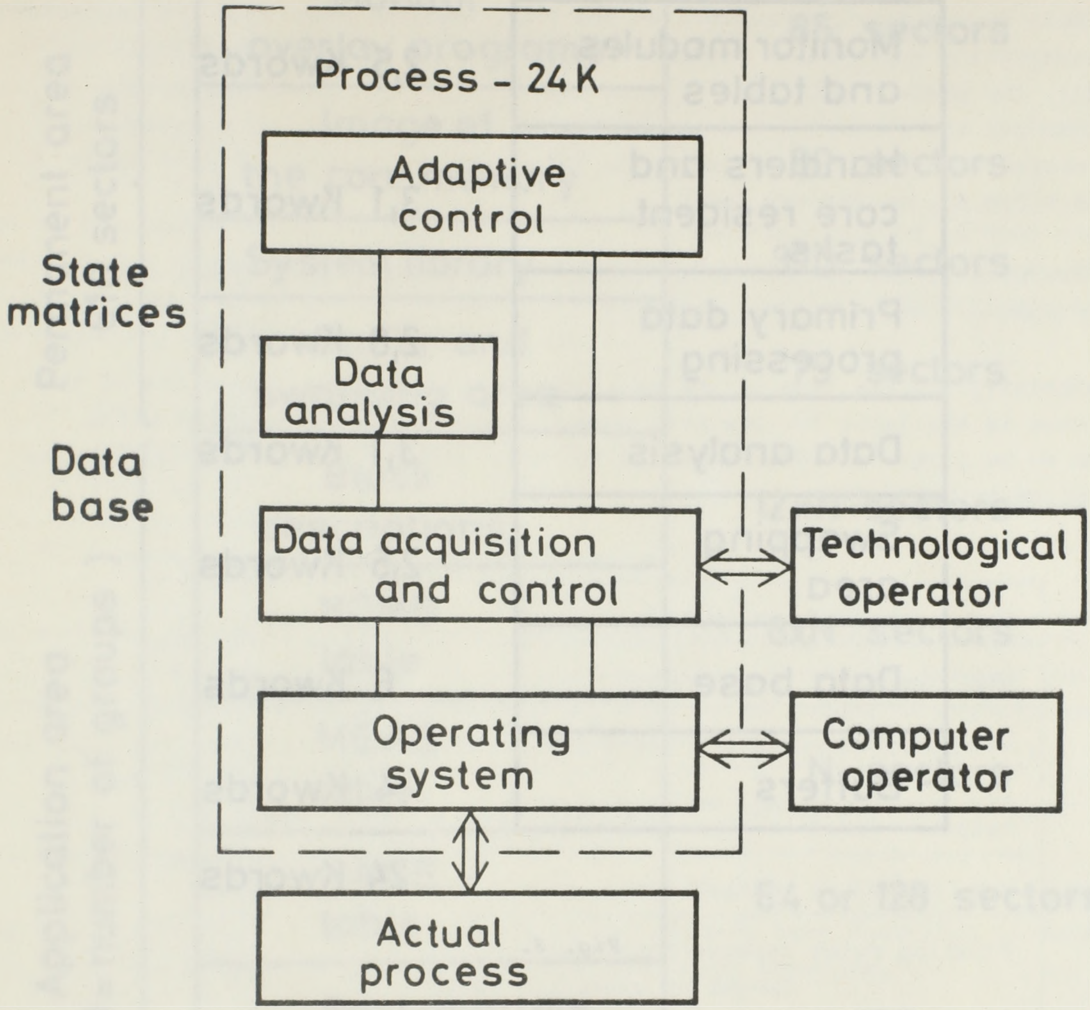


Fig. 2.

Structure of PROCESS-24K.

PROCESS-24K, being a general real-time system, contains only the process independent part of a control system and all of the software aids by which a specific installation can be constructed, for this reason the lower layers are much richer and polished, than the upper ones.

The structure of the core memory and of the disc can be seen in *Fig. 3.* and *Fig. 4.* respectively.The abbrevations used will be defined in CHAPTERS 3. and 4.

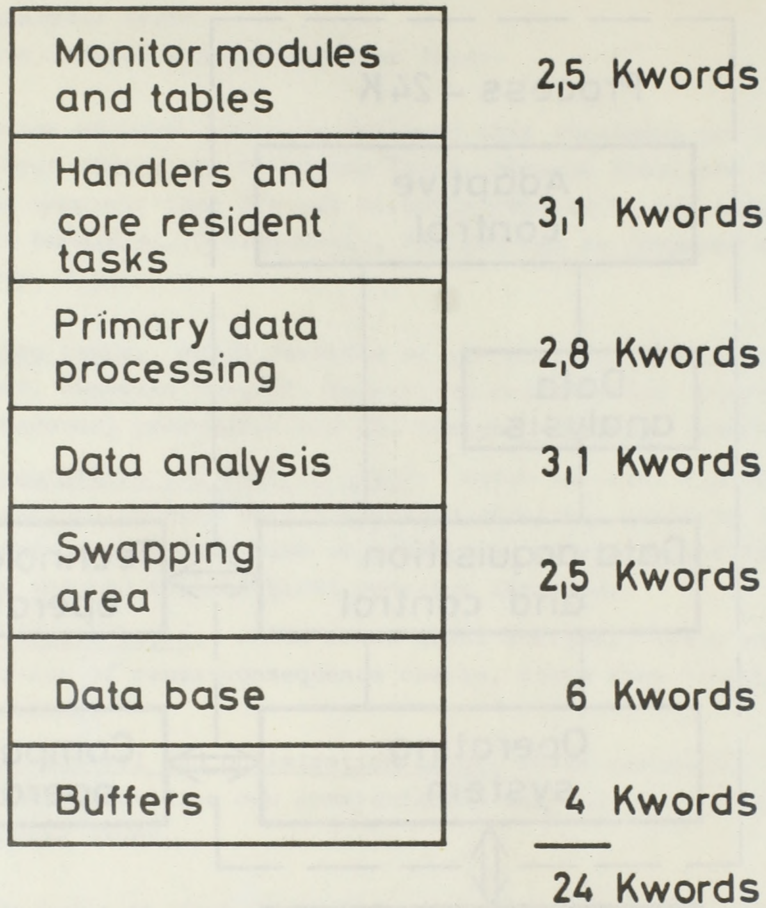


Fig. 3.

Core memory map.

2.3. <u>Throughput of the system</u>

The performance of PROCESS-24K was analysed in a system with 70 analogue variables and with 11 measurements/sec information rate. It was found that the updating of one analogue variable needs 5-6 ms of CPU time. This time includes

- control of multiplexors and A/D converters,
- converting the measured quantity into a floating point number,
- scaling,
- comparison against alarm limits,
- exponential filtering,
- storing in the data base,
- housekeeping of the data acquisition layer.

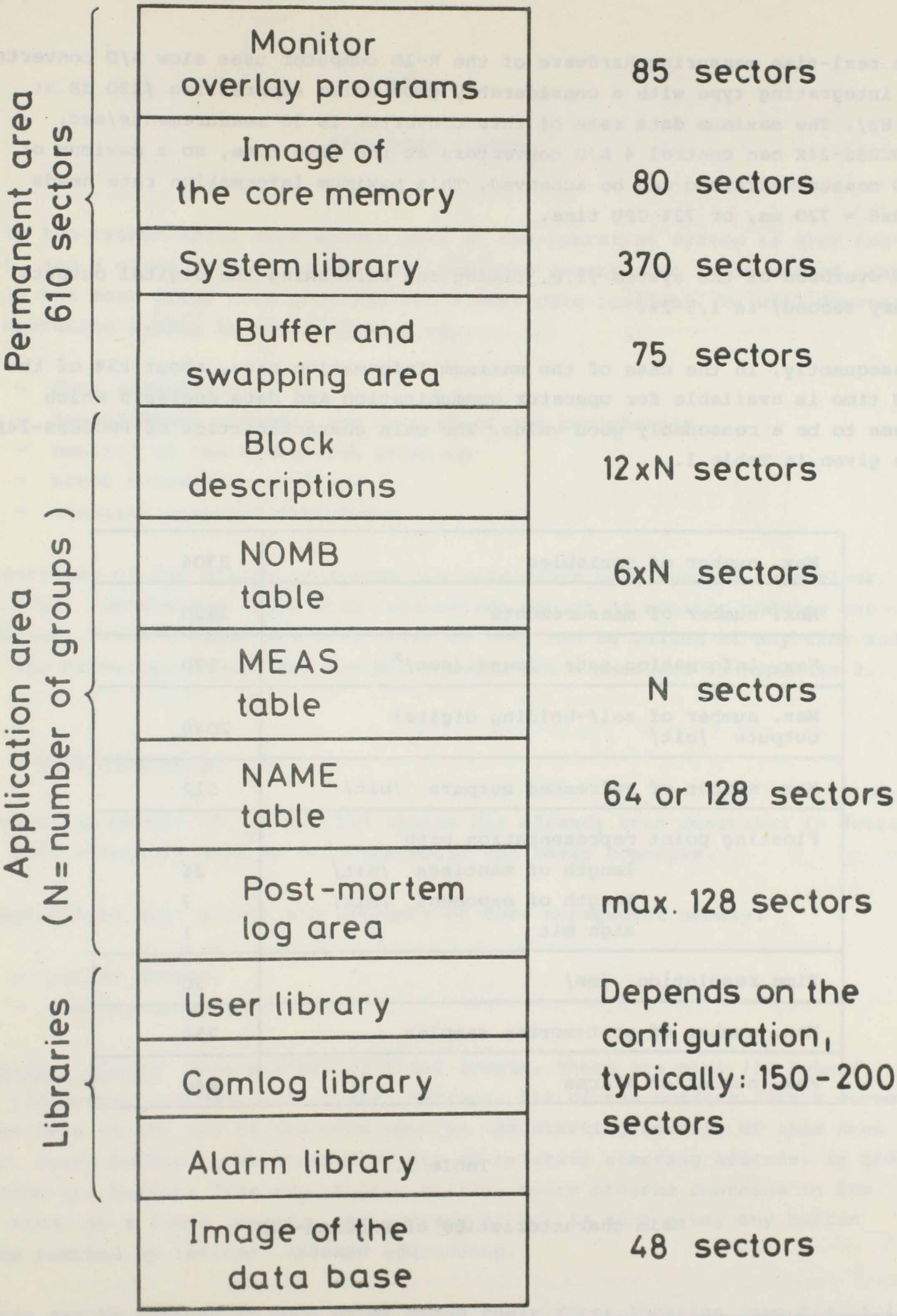


Fig. 4.

Disc map.

The real-time measuring hardware of the R-10 computer uses slow A/D converters of integrating type with a considerably good noise suppression /120 dB at 50 Hz/. The maximum data rate of this converter is 30 measurements/sec. PROCESS-24K can control 4 A/D converters at the same time, so a maximum of 120 measurements/sec can be acheaved. This maximum information rate needs 120x6 = 720 ms, or 72% CPU time.

The overhead of the system /i.e. timing and refreshing the digital outputs every second/ is 1,5-2%.

Consequently, in the case of the maximum information rate, about 25% of the CPU time is available for operator communication and data analysis which seems to be a reasonably good value. The main characteristics of PROCESS-24K are given in Table 1.

Max. number of variables	2304
Max. number of measurements	1920
Max. information rate /meas./sec/[x]	120
Max. number of self-holding digital outputs /bit/	2048
Max. number of refreshed outputs /bit/	512
Floating point representation with length of mantissa /bit/ length of exponent /bit/ sign bit	 24 7 1
Time resolution /ms/	50
Max. number of post-mortem samples	256
Max. number of alarms	640

Table 1.

Main characteristics of PROCESS-24K.

---

x determined by the controlled A/D converters

CHAPTER 3.

OPERATING SYSTEM

Due to the rather small core memory most of the operating system is disc resident and a given part is loaded into the core memory when it is needed. Naturally the most often used programs are always core resident. We will discuss the operating system in the following way:

- core management
- input/output transfer and control of the peripherals
- control of the background programs
- error recovery procedures
- computer operator interface.

The services of the executing system are accessible by a special supervisor call /CSV/ instruction. With this instruction one of 34 monitor modules can be called. These modules are re-entrant so they can be called at any time and from any interrupt level. The monitor modules are summarized in Appendix 2.

3.1. Core management

The core management of the PROCESS system has already been described in detail |22, 23|, therefore here we only summarize the basic concepts.

Two activities fall within the category of core management namely:

- buffer system,
- overlay technique.

The buffer system uses buffers of fixed length. There are mini /10 bytes/, midi /32 bytes/ and maxi /256 bytes/ buffers. All of the buffers form a common buffer area at the end of the core memory; the starting address of this area is ZC. Every buffer is determined by its ZC relative starting address. In order to reach the buffers from any program easily, every program contains in its data area, at a fixed location the address of ZC. In such a way any buffer can be reached by indirect indexed addressing.

Buffers can be chained to each other using their first location /see *Fig. 5.*/. When a buffer has a chained buffer, its first location contains the ZC relative address of the next one, otherwise it is zero.

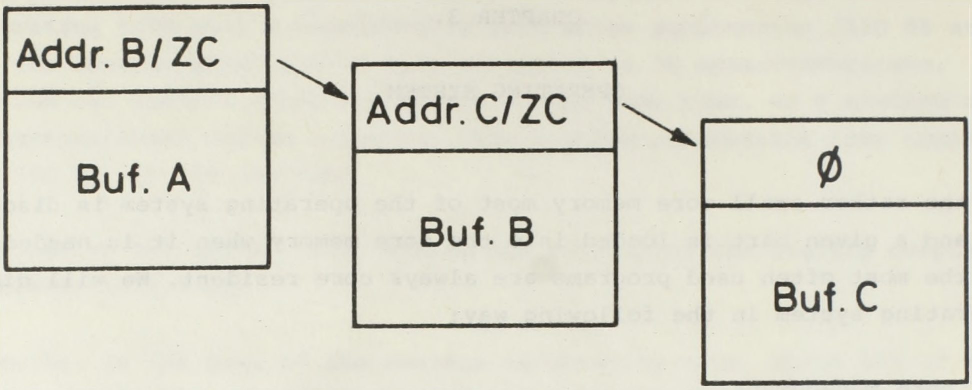


Fig. 5.

Chaining in the buffer system.

The service of the buffer system is reached by 4 supervisor modules, three of them reserve a mini-, midi- or a maxi buffer /MINI, MIDI, MAXI/ respectively while the fourth releases a buffer /FREE/. When a program reaches an EXIT instruction, or it is aborted due to an error, all of its reserved buffers are freed automatically. The monitor checks the number of the buffers allocated to a user program and if it is greater than a predetermined value, the user program is aborted and its buffers are released.

The applied overlay technique is very simple but highly effective. It presumes

- every section using overlay procedure is not longer than 256 byte /i.e. 1 disc sector/
- when a program runs, only one of its sections is in the core
- when a program waits for the execution of an I/O transfer, none of its sections is in the core
- sections are not written back to the disc, common variables must be stored in the core resident root of the program
- from every section it is obligatory to return to its calling section but the EXIT instruction can be executed in any section /see *Fig. 6.*/.

These constraints are sometimes rather strict but the primary aim is to occupy as small memory space as possible. In such a way the overlay sections run in maxi buffers which results a very simple solution.

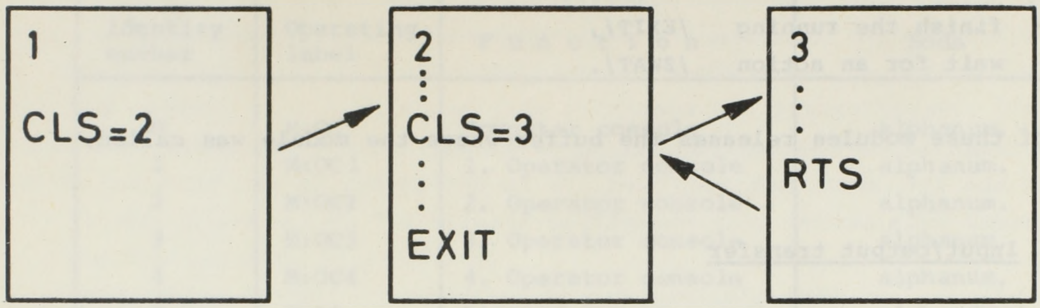


Fig. 6.

Overlay technique in the PROCESS system.

Every section must begin with the page relative starting address in its first word /*Fig. 7.*/.

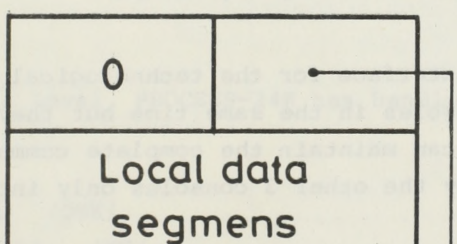


Fig. 7.

Structure of an overlay section.

Four monitor modules form an interface between the programs and the operating system, which can

- call an overlay section /CLS/,
- return from a section /RTS/,
- finish the running /EXIT/,
- wait for an action /ZWAT/.

Each of these modules releases the buffer where the module was called.

3.2. Input/output transfer

The handling of all conventional peripherals is quite the same as in other R-10 /MITRA/ operating systems |24, 25|. The operating system distinguishes logical and physical peripherals. Every program uses logical peripherals while the actual data transfer is carried out through a physical one assigned to the given logical peripheral. This solution is very flexible since if, for example an error occurs in a peripheral unit, the computer operator can assign very easily another one, to replace the faulty equipment.

The logical peripherals are the following /Table 2./:

Computer console is a typewriter by which the computer operator can instruct the system, and where the errors of the computing hardware and the programming errors are reported.

Operator console is an interface for the technological operator. The system can handle 4 operator consoles in the same time but they are not identical. Through OC1 the operator can maintain the complete communication with the system /see 4.7./ while by the other 3 consoles only interrogating is possible.

Listing output produces different types of lists.

Listing log is used for typing different types of logs.

Backing memory provides a large backing store /typically magnetic tape/.

Synchronous connection is planed to maintain a data link with other computers.

Elementary input is a data input into the system.

Elementary output is a data output of the system.

System disc is an area of the disc dedicated to the users. User programs can communicate only with this given area, they have no access to the whole area of the disc.

Identity number	Operating label	F u n c t i o n	Mode
0	M:CC	Computer console	alphanum.
1	M:OC1	1. Operator console	alphanum.
2	M:OC2	2. Operator console	alphanum.
3	M:OC3	3. Operator console	alphanum.
4	M:OC4	4. Operator console	alphanum.
5	M:LO	Listing output	alphanum.
6	M:LL	Listing log	alphanum.
7	M:MBG	Backing memory	bin.,alphanum.
8	M:CLS	Synchr.connection	bin.,alphanum.
9	M:EI	Elementary input	bin.,alphanum.
10	M:EO	Elementary output	bin.,alphanum.
11	M:SY	System disc	binary

Table 2.

Logical peripherals.

At physical peripheral's level, PROCESS-24K has handlers for the following equipment:

- fixed head disc /DSK/,
- magnetic tape unit /MT/,
- paper tape reader /PTR/,
- paper tape punch /PTP/,
- asynchronous data line /CLA/ for logging typewriter,
- Teletype /ASR/,
- alphanumeric display multiplexor /MPX/,
- line printer /LPT/,
- synchronous data link /CLS/.

These handlers are special programs at different interrupt levels. The assignment of logical and physical peripherals can be seen in Table 3.

In this system there are four monitor modules organizing the input/output data transfer, viz.

- ZIO - input/output communication
- ZTYP - combined output transfer with input
- ZWAT - waiting for the execution of an I/O transfer
- ZDIO - disc transfer.

The parameters of each module has to be given in a control block /CB/ and due to the intensive swapping used, it is obligatory to put both the CB and the actual data buffer into the buffer area.

Operating label	Standard assignment	Other peripherals can be used
M:CC	ASR	-
M:OC	MPX	CLA, ASR, NO
M:LO	LPT	CLA, ASR, MPX, PTP, MT, CLS, NO
M:LL	CLA	LTP, ASR, MPX PTP, MT, CLS, NO
M:MBG	MT	NO
M:CLS	CLS	NO
M:EI	PTR	ASR, NO
M:EO	PTP	ASR, NO
M:SY	DSK	-

Table 3.

Assignment of logical and physical peripherals.

With a control block the transfer of max. 256 bytes can be specified except for the M:SY peripheral, where this amount can be much longer. If more then 256 bytes has to be sent/received, a chain of CB-s can be used.

Only one dedicated area of the disc /System disc, M:SY/ can be reached by the ZIO module. It is obvious that the system programs have to reach the whole disc available, for this purpose the ZDIO module can be used. This module was developed exclusively for system programs, its use is forbidden to users.

3.3. Background organization

At maximum data rate PROCESS-24K has about 20-25% free CPU time /see Section 2.3./ which is a very nice amount for background programs. In this context a background program is a code whose time relations are insignificant, its execution is not urgent. There are two types of background programs:

- codes connected to the real-time tasks, called COMLOG /COMputing or LOG producing/ programs,
- utility programs, having no direct connections with process control.

Clearly, the former group is more important from a system operation point of view, therefore the system handles them with a higher priority. Both type of programs run at the lowest interrupt level. If two background programs request running concurrently, the scheduler will choose that one which has real-time relations, otherwise it will decide on a first-in-first-out base. COMLOG programs can be initiated by other real-time programs but some of them can be started by the technological operator as well, in contrast utility programs can be called only by the operator. The space in the core memory available for background computing is 5 Kbytes. When a program is longer than this swapping area, it has to be partitioned. A partition of a program can call its continuation by the M:LOAD monitor module. The partitions can communicate with each other at the end of the swapping area because the M:LOAD module loads only the actual length of the called program i.e. it does not use more territory than necessary.

The swapping of the background programs has already been discussed in detail in the literature |12, 13|.

The background programming is supported by 2 monitor modules:

BIBL - gives the starting sector and the actual length /in sectors/ of a file determined by its 6-character long name and a flag byte specifying the library

81	-	system library
01	-	user library, executable program
02	-	user library, assembler source
04	-	user library, logsheet source
08	-	user library, PROCESS source
10	-	user library, generating data
40	-	alarm tree library

LOAD - loads a program determined by its starting sector and length into the swapping area.

The system library contains the usual service programs for editing, compiling, loading, mapping as follows:

- text editor (/TEXTE)
- compilers:

assembler	&BSATR
PROCESS compiler	*AUTOC
logsheet compiler	'LOGTR

- loaders:

assembler loader	+BSALD
task loader	.TASLK
COMLOG loader	=COMLK
PROCESS loader	@LINK
logsheet loader	"LOGNL

- memory and disc dump	MDMAP
- post-mortem log dump	PMLIST
- system generator	$GENES
- alarm tree generator	ALGEN
- refreshed output generator	TDIGO
- mapping the actual groups	GRSN
- mapping the variable of a group	CHSN

The actual functions of these programs will be discussed in Chapter 7. The length of the system library is 380 sectors, the length of the user and COMLOG library are determined at the system generation.

3.4. Error recovery procedures

In a real-time system an error may happen at any time and it is highly important to maintain the primary function of the system. For this reason the operating system contains procedures to avoid

- errors caused by the environment,
- faults, generated by the programs.

The environmental errors may be caused

- either by the power supply,
- or by the computing hardware.

When the electric power fails, the CPU generates an interrupt at level 31, which results a master clear. When the power returns, the CPU generates an interrupt at level 30, by which a PWUP is initiated. This program

- indicates in a flag /FL:UP/ that this running is not an initial program loading /IPL/ but a recovery procedure,
- calls a DBØOT program which loads the image of the core, from the disc, afterwards the STARTER program is initiated.

This STARTER determines whether it is an IPL phase or an error recovery. In the latter case the data base and the time registers of the system are not written over, so the system continues its operation. At the end of this starting procedure a message it typed out at the computer console, registering the time of the power failure.

The error of the computing hardware can be catastrophic or may only cause a degradation in the system performance. When a fundamental part of the hardware goes wrong /central processor, disc/, not too much can be done; the CPU halts and a light on the front panel of the computer indicates the type of error /e.g. memory parity error, disc error/. When a peripheral unit goes wrong, the computer operator can either replace the erroneous equipment with a spare part or can assign its function to another unit. For this purpose the operating system measures the time of every data transfer and if this time is longer than 2 minutes, it sends a time-out message to the computer console.

It is more difficult to avoid a programming error. The R-10 computer has two operating modes: master- and slave mode. In the slave mode a program uses only a subset of the operation repertoir of the computer, in this case the use of the "most dangerous" instructions is forbidden. User programs can use the computer only in slave mode. When a user program

- uses forbidden instruction,
- wants to write into the system memory area,
- uses a non existent instruction

the CPU refuses to execute the given operation and calls the monitor module Ø /System Trap/. This module

- aborts the erroneous program
- types out an "abort report" on the computer console.

3.5. Computer operator interface

When the computer operator wants to instruct the operating system, he has to cause an interrupt at level 5, by pressing a pushbutton either on the front panel of the CPU or on the computer console. In this case, the system types out the actual time and waits for the instruction, which can be one of the following:

CALL - with this instruction a utility program either from the system library or from the user library can be loaded into the background area and then started.

ASSIGN - this instruction assigns a physical peripheral to a logical one.

Y:BG - aborts the running background program.

TIME CORRECTION - changes the actual date /day, hour, minute/ in the system.

All these communications are carried out through the computer console.

CHAPTER 4.

DATA ACQUISITION LAYER

The fundamental functions of this layer are to reflect the actual process in the data base of the system, to produce proper answers to every change in the "outer world" and to inform the technological operators about the present situation. At this level the problems of each process are very similar to each other so a general solution to these problems can be given and the user has to fit this solution to his actual environment. For example, every process control system needs measuring of the process, so the general solution to this problem is to organize and to carry out these measurements but the user has

- to specify the cycle of a given measurement,
- to allocate an input route for it,
- to specify the gain of amplifier etc.

For this reason PROCESS-24K has table controlled subsystems: measuring, data processing, displaying, logging and the user has to specify their operation only. This specification is performed by filling out control tables of the particular subsystem by means of special utility programs.

Since PROCESS-24K is a measurement oriented system, its real-time tasks are organized according to the measuring cycle time of these tasks. All measuring and/or control tasks with the same cycle time form a group. Each group is represented by its Group NumBer /GNB/. PROCESS-24K can contain a maximum of 48 groups, numbered from ØØ to 47.

Each group may contain max. 48 variables. In a group every variable is represented by a Block NumBer /BNB/, so each variable can be identified by a 4-decimal identity number /GNB, BNB/. For the operator's convenience a max. 6-character long mnemonic is associated with every identity code so a variable can be called by its name instead of a 4-decimal long number.

Since PROCESS-24K can contain max. 48 groups and each group has max. 48 variables, this system can handle max. 48x48 = 2304 variables.

4.1. Data base structure

Each variable has

- an actual value
- a flag byte reflecting the actual state of the measurement /e.g. invalid, overlimit, etc./, and
- auxiliary information /e.g. name, dimension, absolute values of different type/.

All of this information forms the data base of the system. Whereas the first two parts have to be up-dated in every measuring cycle, the third part is constant. For this reason rapid access is needed to the first two parts, but as the constant part is needed only occasionally, the first two parts are stored in the core memory, the constant part is on the disc.

The core resident data base reflects the group structure of the PROCESS system. Each group has a 256-byte long page in the operating memory where the values and the flags of its variables are stored. The structure of the core resident data base of a group is given in *Fig. 8*.

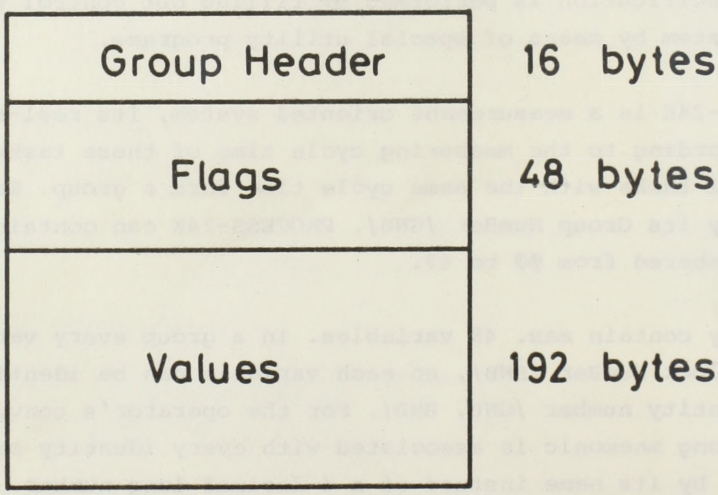


Fig. 8.

Data base structure of a group.

This page is divided into 3 parts, namely

- a group header
- a flag field
- a value field.

The group header contains the following information

- the group number,
- the measuring cycle time,
- the identity code of the coupled COMLOG program /if there is any/,
- data for the post-mortem log /starting sector, actual pointer etc./,
- data of the loaded blocks.

The flag field consists of 48 flag bytes. Each byte reflects the present state of a variable, the meaning of the individual bits are the following:

operating bit - if this bit is zero, the given variable is not measured/controlled

analysis bit - this bit is reserved for the data analysis layer in order to reflect, e.g. if a given variable is used in the alarm analysis or not

digital control bit - if this bit is zero, control functions with this variable are not carried out, only measurement tasks

competency bit - this bit reflects whether the system needs a competence checking when the operator tries to alter, e.g. limits of the variable /see 4.7.1./

lower limit - it specifies the overlimit, if this bit is one, the lower alarm limit is passed

validity bits - with the following structure

OO	-	valid
10	-	invalid measurement at the first time
11	-	invalid
O1	-	take it as valid

overlimit bit - if this bit is one,the variable passed its upper or lower alarm limit, specified by bit 4.

The structure of the flag byte is given in *Fig. 9*.

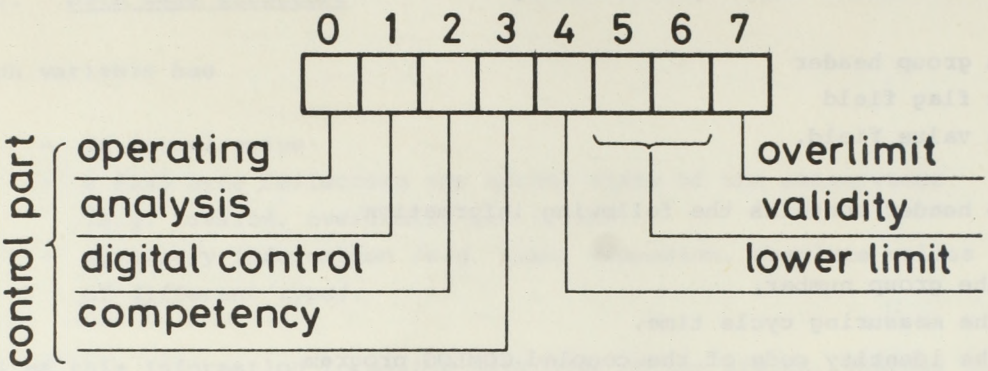


Fig. 9.

Flag byte structure.

The value field can store up to 48 four-byte long values. The structure of an analogue variable is given in *Fig. 10.* In this floating point representation a N number in the

$$0{,}6909 \times 10^{-76} < N < 0{,}7231 \times 10^{76}$$

range can be represented with a 6 digit accuracy, and the value of a variable is given with the expression

$$N = M \times 16^{c-64}$$

In the case of a digital variable, only the first 2 bytes of the value are used, the content of the following 2 bytes are insignificant.

PROCESS-24K contains a 48-page long data base in the core, moreover the image of this area is on the disc with the initial values and flags of the variables. At the IPL phase this disc area is also loaded into the core with the initial quantities in order to define the starting state of the control system.

The disc resident auxiliary part of the data base form two tables /NAME and NOMB/ on the disc. The NAME table serves for associating an identity number /GNB, BNB/ to a mnemonic name of a variable. A 8 byte-long item of this table is given in *Fig. 11.*

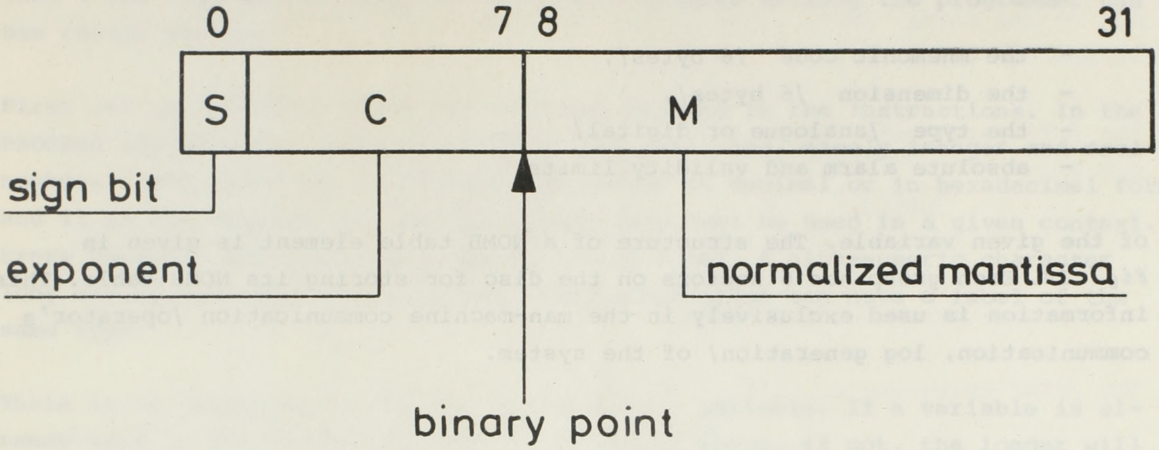


Fig. 10.

The structure of an analogue variable.

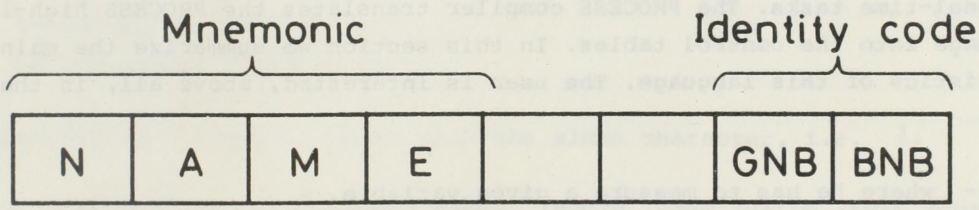


Fig. 11.

NAME table element.

A special supervisor module /NAME/ has access to this table, by which an identity code can be associated with a given mnemonic.For the operator's convenience, there is an operator command to call the identity code /see Appendix 5./ of any variable.

The NOMB table is organized according to the identity codes instead of the mnemonics.Every element of that table is 32 bytes long and stores

- the mnemonic code /6 bytes/,
- the dimension /6 bytes/,
- the type /analogue or digital/
- absolute alarm and validity limits

of the given variable. The structure of a NOMB table element is given in *Fig. 14.* Each group has 6 sectors on the disc for storing its NOMB table. This information is used exclusively in the man-machine communication /operator's communication, log generation/ of the system.

4.2. The PROCESS language

It was mentioned in the introduction of CHAPTER 4 that PROCESS-24K has table-controlled subsystems and the user only has to specify their operation, moreover this specification is done by filling out various types of control table. The generation of these control tables is carried out by a system program / @LINK/, which loads into the system the information produced by the PROCESS compiler. A fundamental feature of this loader is its real-time operation, i.e. it loads a program into a running system without interferring with the real-time tasks. The PROCESS compiler translates the PROCESS high-level language into the control tables. In this section we summarize the main characteristics of this language. The user is interested, above all, in the following:

- where he has to measure a given variable,
- what he has to do with the measured quantity,
- which control valve must be activated at a given time.

He is uninterested in the inner structure of the PROCESS-24K, in the operation of the different routines, in the content of the control tables. One thing is important for him, to write down his problem as simply as possible. In the construction of the PROCESS language the user's viewpoint was taken to be fundamental. For this reason the basic concept for the user is a channel. There are two types of channels, viz. a measurement with its processing or a control action. All tasks of a given channel will be considered as channel program. The compiler breaks the channel program into blocks and generates data for the different control tables. At a given time only one channel program can be translated or loaded into the system.

A channel program is a series of statements. Every statement must be written into a new line and in order to facilitate program writing the programmer can use coding sheets.

First let us look over which operands can be used in the instructions. In the PROCESS language two types of constant are permitted, namely integer and real numbers. Both types can be represented either in decimal or in hexadecimal form and it is the compiler who decides which form must be used in a given context. Every variable has a symbolic name which is a max. 6 alphanumeric character long string starting with a letter. Each instruction can have a label of the same type.

There is no necessity to declare the type of a variable. If a variable is already used in the loaded program, it is stored there, if not, the loader will assign a space for it. The type of variable is also determined by the compiler on the basis of its occurence. Variables and numbers, connected to each other by the four rules of arithmetic form an expression. The use of parentheses is not permitted; the expressions are interpreted from the left to the right. The length of an expression is limited only by the length of the line. The type of expression is also defined by the compiler.

A PROCESS statement can be

- either declarative,
- or executable instruction.

Every declarative statement begins with the slash character, i.e. /.

There are only 6 statements of this type: /BEGIN,/STRT,/VALUE,/ANAL,/GUARD,/END. The first five can be used only at the beginning of a channel program while the last indicates the end of a program.

The /BEGIN statement must be the first instruction of a program. In this statement the user can specify

- the names of the channel and of its variable,
- the initial value of the variable,
- the dimension of the variable.

This statement gives information for the NAME and NOMB tables.
The initial data base is determined by the /VALUE,/ANAL,/GUARD statements.
The /VALUE declaration gives a starting value to the variable. The argument of the statement is a 4-digit hexadecimal number /in the case of digital variables/ or a 6-character decimal number /in the case of analogue variables/.

The analysis and the competency bits of the flag can be set by the /ANAL and the /GUARD statements respectively.
All of the former three declarations can be omitted, in this case the initial values are zero.

The /STRT statement defines where the channel program is initiated from. This can be

- the input of an A/D converter,
- a digital input, or
- another channel program.

The /STRT statement determines the information for the MEAS table.

There are two types of executable statements. In the first set, the order of the execution is predetermined; the compiler generates from these statements the fixed format blocks of the PROC table. The order of the statements corresponds to the usual order of the operations in the process control problems and equals the sequence of the instructions in the fixed format blocks. This order is given in *Fig. 12.* /The instruction set of the PROCESS language is summarized in Appendix 3./

The /BEGIN, /STRT and the first executable statement determine the type of block generated by the compiler.

In the second set, the order of the statements is free from the viewpoint of the language, these instructions form the free format blocks. The language has

- arithmetic,
- logical,
- data moving,
- branching,
- control,
- function generator,
- program transfer instructions.

In the arithmetic subset of the statements there are addition, subtraction, multiplication, division and power functions. Every instruction has four types because the operands are the content of the E, A accumulators and

- a parameter,
- the variable of the given block,
- the variable of another block,
- or a value stored at a given address.

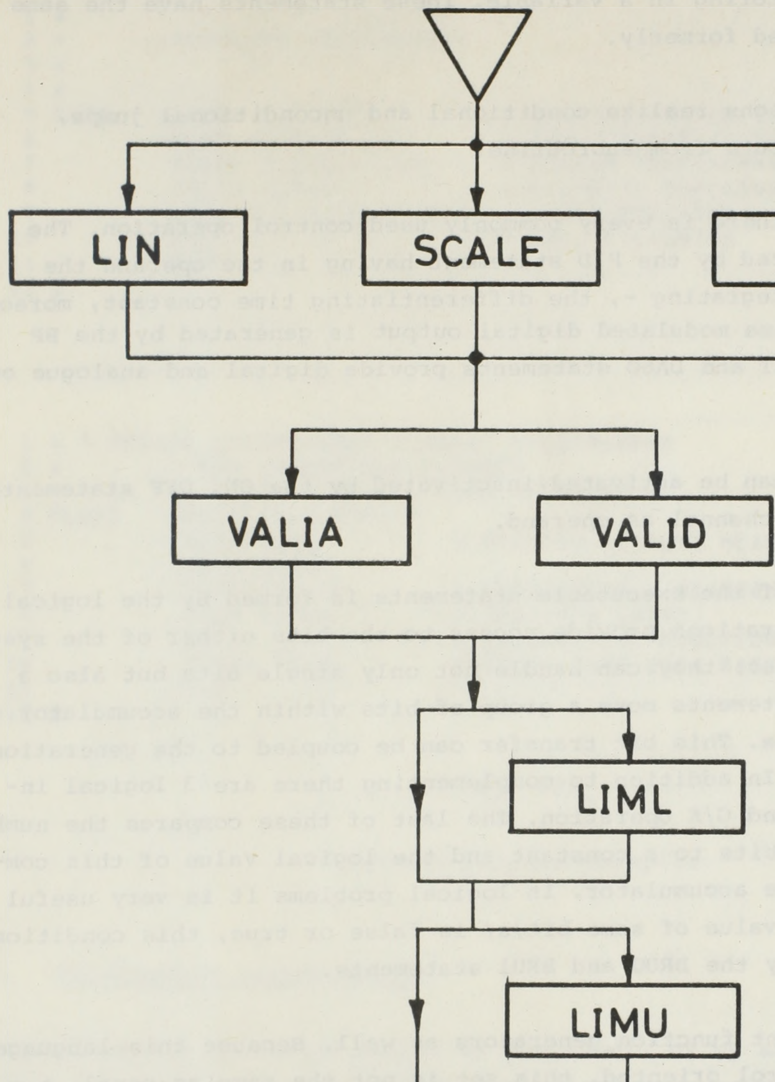


Fig. 12.

Order of the execution in the fixed format channel programs.

The data moving instructions serve for loading the value of a variable into the accumulators, or storing in a variable. These statements have the same types as the ones listed formerly.

The branching instructions realize conditional and unconditional jumps, subroutine call and return from subroutine.

In the control subset there is every commonly used control operation. The PID algorithm is realized by the PID statement having in the operand the proportional -, the integrating -, the differentiating time constant, moreover the set point value. Time modulated digital output is generated by the BP statement, while the BIT and DA60 statements provide digital and analogue output, respectively.

Every channel program can be activated/inactivated by the ON, OFF statements having the name of the channel as operand.

A rather large subset of the executable statements is formed by the logical instructions. These operations provide access to the bits either of the system, or of the group variables; they can handle not only single bits but also a group of bits. Some statements move a group of bits within the accumulator or into a selected variable. This bit transfer can be coupled to the generation of the complement too. In addition to complementing there are 3 logical instructions: NAND, NOR and G/K operation. The last of these compares the number of ones in a series of bits to a constant and the logical value of this comparison is stored in the accumulator. In logical problems it is very useful to jump if the logical value of some bit/s/ is false or true, this conditional branching is realized by the BRU0 and BRU1 statements.

PROCESS-24K has different function generators as well. Because this language is measurement and control oriented, this set is not the same as usual, e.g. in FORTRAN. There are exponential-, natural logarithm, square-, square root extension-, integer-, absolute value function generators, but there is no sine or cosine function for they are not frequently used in calculating a control signal. Instead of these there are special function as RANGE which generates $(\sqrt{10})^x$ for evaluating the multiplying constant of a measuring range or FLOAT which converts a logical variable into a real number which is usual for calculating the output of a digital/analogue converter. Naturally there is a possibility to define external functions too, so the user can also produce new function generators. Every generator gets the variable, and provides the new value in the E, A accumulators.

In this paper we do not seek for completeness, our only endeavour was to present the main features of the PROCESS language. A short description of the instructions is given in Appendix 3. Program examples can be seen in *Fig. 13.*

```
*
*       PREASURE MEASUREMENT
*
*
PPR1   /BEGIN 1,A,ATT            * DEFFINITION OF THE VARIABLE
       /STRT AD10,13,100         * INPUT SPECIFICATION
        SCALE 5,115,-1,023       * LINEAR TRANSFORMATION
        VALIA 0,5,5              * VALIDITY CHECKING
        LIML 1,6,0,05,(1,2)      * ALARM CHECKING
        FILT 0                   * EXP, FILTERING
       /END
@
```

```
* 7 DECADE LOGARITHMIC CURRENT MEASUREMENT
*          FOR IONIZATION CHAMBER
*    IN THE RANGE 0,0001 - 1000 MIKROAMP
FLUX3   /BEGIN 1,A,MIKAMP
        /STRT FLX3              * STARTED BY FLX3 MEASUREMENT
         SETN FLX3
         EXTRS 57,14            * REPRESENTS 1 MIKROAMP,
         DIVS 6,204             * 100/(7 X LN10)
         EXP                    * EXPONENTIAL FUNCTION
         STORN FLUX3            * STORE INTO DATA BASE
        /END
@
```

Fig. 13.

PROCESS program examples.

4.3. Measurement organization

Measurements are initiated either by the timer, or by an interruption. In this section only the timer controlled, periodical measurements are discussed, the interrupt initiated actions will be treated in Section 6.1.

The measuring subsystem has a base time, T, which has to be an integer multiple of 50 ms. In general 1 sec is preferred. PROCESS-24K can handle 8 different measuring periods with

$$T_i = 2^i T \qquad /i = 0, 1, \ldots, 7/$$

cycle times.

The groups with the same cycle time form a measuring chain. When the measurement of a group is finished, the system searches for another group with the smallest cycle time to measure. If this subsystem has no more groups to be measured, it waits for the next initiation.

Competence	Type	2 bytes
Mnemonic		6 bytes
Dimension		6 bytes
Absolute alarm and validity limits		16 bytes
Reserve		2 bytes

Fig. 14.

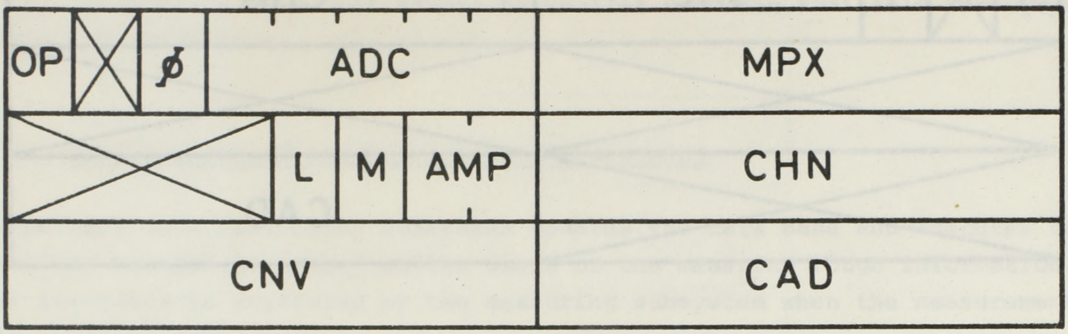
NOMB table element.

In every group the measurements are organized into chains according to the source of the information. There are 5 chains, one digital and four analogue ones because the A/D controller can handle 4 A/D converters at the same time. The measurements of these five chains are executed parallely which saves a considerable amount of time.

Each measurement is specified by a 6-byte long description. This description is given in *Figs. 15.* and *16.* for analogue and for digital measurements respectively.

Every group has a 256-byte long MEAS table containing max. 40 measurement descriptions and a header which stores the following information:

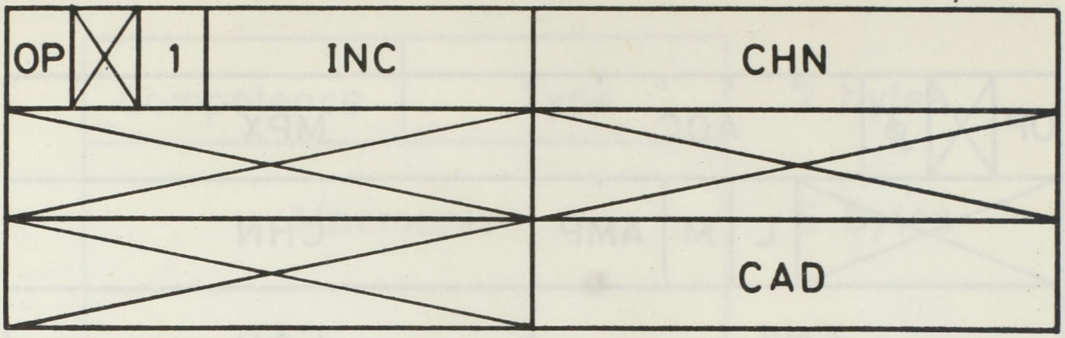
- group number,
- post-mortem data,
- 5 pointer pairs, indicating the first and the last element of each measuring chain.



where OP - operating bit
ADC - address of the A/D controller
MPX - address of the multiplexor
L - last specification, this bit instructs the A/D controller to execute the measurements specified
M - measurement, if this bit is zero, then the specified action is only a test instead of measurement
AMP - amplification, it specifies the gain desired (1, 100, 250, 1000) of the amplifier
CHN - channel address in the multiplexor
CNV - address of the A/D converter
CAD - chain address for the next block description

Fig. 15.

Measurement description of the analogue channels.



where OP - operating bit
INC - address of the digital input controller
CHN - channel address
CAD - chain address for the next block description

Fig. 16.

Measurement description of the digital channels.

The MEAS tables are stored on the disc, and the measuring subsystem loads the appropriate table into the core when it is needed and operates according to its control. The operation is driven by the interrupt of the A/D controller, indicating the execution of the A/D conversion.

The result of a measurement is either a 12-bit long pattern in the case of analogue channels, or a 16-bit long pattern in the case of digital channels. Moreover each measurement is specified by a 3-bit long status word which indicates

- whether the execution of the measurement was erroneous or not,
- whether there is an overflow in the A/D converter or not,
- the type /analogue or digital/ of the measurement.

When the measurements of a group are completed, this rough information is passed to the primary data processing subsystem.

When designing a measuring subsystem, it has to be kept in mind that the actual information rate must always be smaller then the available throughput of the A/D converters.

4.4. Organization of the primary data processing

The primary data processing subsystem updates the data base and executes the specified control functions on the basis of the measured rough information. This subsystem is initiated by the measuring subsystem when the measurement of a group is completed.

The organization of the primary data handling is very similar to that of the measuring subsystem, its operation is controlled by the so called PROC tables. The description of the primary data processing, like the measurement description, is block oriented. Every variable in a group has one or more blocks in the PROC table of the group. The length of a block is 32 bytes. There are two types of blocks, namely

- fixed format,
- free format blocks.

The fixed format blocks serve for evaluating the value of a variable from the measured rough information. In this block type the order of the actions are predetermined e.g. if somebody wishes to perform validity checking, this has to be done before executing an alarm limit cheking. The actions needed are stored in an Instruction Word /IW/, whose every bit corresponds to a given action; the order of execution is given in the order of the bits of this word. Moreover there are actions which can only be used by themselves, e.g. one can prescribe either a linear transformation, or a linearization with a predetermined characteristic but both can not be used together. The parameters of the actions are stored in the block.

In the free format blocks the order of the instructions are not predetermined, an instruction is represented by a code following its parameter/s/, so they are instructions of variable length.

The structure of the blocks is given in *Fig. 17*.

If the processing of a variable can not be described in one block a further block can be chained to the given block by the CAD word. In a PROC table there is room for 80 blocks which, in general, is sufficient.

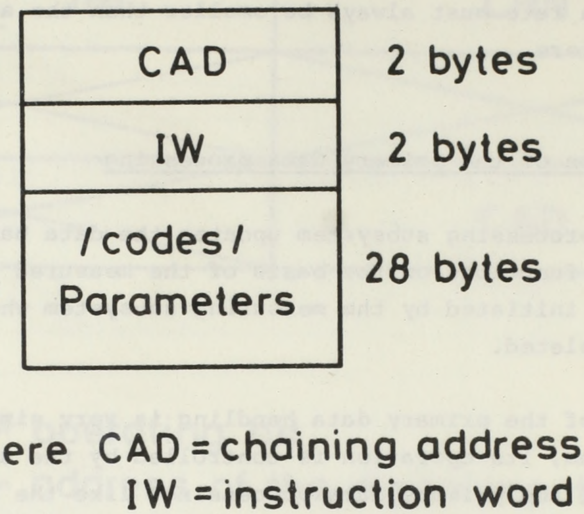


Fig. 17.

Block structure of primary processing.

The primary processing is executed by an interpretative processor i.e. this subsystem interprets either the instruction word /fixed format/ or the operation codes /free format/ of a given block and calls the proper subroutine with the parameters stored in the block. The available subroutines are summarized in Appendix 3.

The compiler of the PROCESS high-level language fills up the appropriate MEAS and PROC tables. /See Section 4.2./

If a problem can not be described in the PROCESS language, assembler programming is also permitted in the primary processing level. Every group can have an assembler task which is executed after the interpretative processing. Each task consists of 3 parts:

- a task header,
- a task data segment,
- a task program segment.

The task header is 16 bytes long and contains the following information:

- a semaphore byte, reflecting the state of the task /i.e.: inactive, waiting etc./
- starting address
- 4 working cells for supervisor modules
- a 6-byte long task name.

Since a primary processing task runs at a high priority level it must be a short program; thus the max. length of a task is limited to 256 bytes /1 sector/. The structure of a task is given in *Fig. 18.*

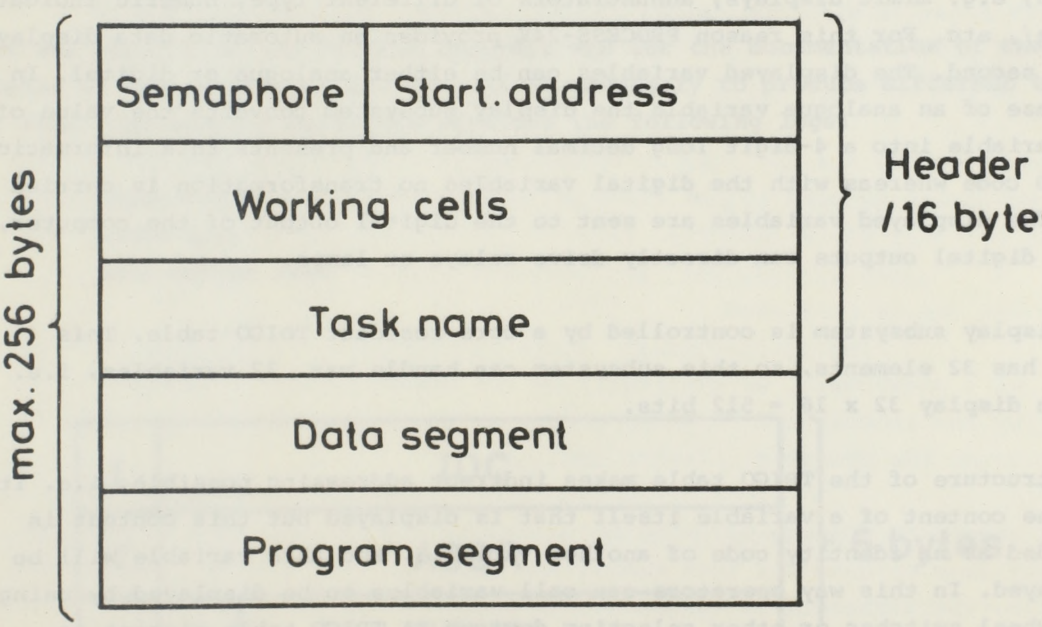


Fig. 18.

Structure of a primary processing task.

When the primary processing of a group is finished the processing subsystem can initiate a COMLOG program which will run in the background. The length of this program is not limited but due to its lowest priority level, it is executed rather slowly.

The description of the primary processing of the groups are stored on the disc and the proper area is loaded into the core when the primary processing subsystem begins to work on the rough information of a given group. This disc area consists of 12 sectors, namely

- 5 sectors for fixed format block descriptions
- 5 sectors for free format block descriptions
- 1 sector for the primary processing task
- 1 sector for external functions.

In order to save time the processing subsystem checks whether the needed area is in the core and it swaps with the disc only when a new area is called.

4.5. Automatic display functions

In a control room a considerable amount of different data presentation are needed, e.g. mimic displays, annunciators of different type, numeric indicators /Nixie/, etc. For this reason PROCESS-24K provides an automatic data display every second. The displayed variables can be either analogue or digital. In the case of an analogue variable the display subsystem converts the value of the variable into a 4-digit long decimal number and presents this information in BCD code whereas with the digital variables no transformation is carried out. The displayed variables are sent to the digital output of the computer, these digital outputs can directly drive relays or lamps.

The display subsystem is controlled by a core resident TDIGO table. This table has 32 elements, so this subsystem can handle max. 32 variables, i.e. it can display 32 x 16 = 512 bits.

The structure of the TDIGO table makes indirect addressing possible, i.e. it is not the content of a variable itself that is displayed but this content is regarded as an identity code of another variable, and this variable will be displayed. In this way operators can call variables to be displayed by using thumbwheel switches or other selecting devices. A TDIGO table element is given in *Fig. 19.*

The abbreviations used in *Fig. 19.* are

IDC - identity code of the variable in BCD code

I - indirect address, if

I = 0, IDC specifies the variable to be displayed

I = 1, IDC specifies a digital variable storing the identity code of the variable to be displayed.

ADDR - address of the digital output line

T - type of variable /T = O digital, T = 1 analogue/

V - variable decimal point, if

V = 1 the position of the decimal point is fixed

V = O the position of the decimal point is moving, it is specified by TPNT and P

TPNT - identity code of the variable storing the decimal point and the sign of the displayed analogue variable

P - indicates whether the upper or the lower byte is used in the decimal point representing variable.

The TDIGO service program is designed to fulfil the TDIGO table of PROCESS-24K.

4.6. System log functions

For the purpose of the further analysis and for the documentation of the events of the controlled process, it is necessary to produce different types of logs. The logging subsystem provides the following logs:

- event log,
- plant log,
- post-mortem log.

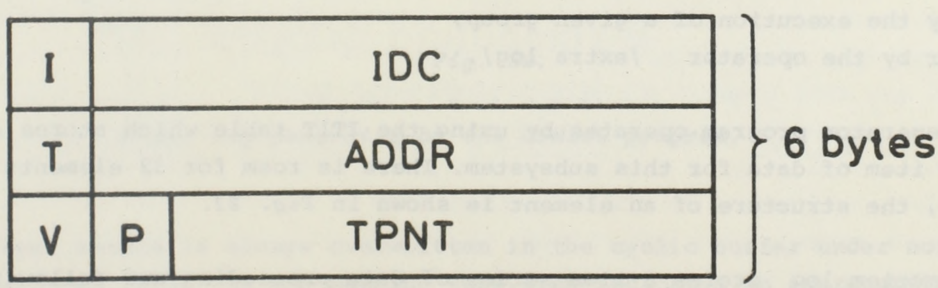


Fig. 19.

TDIGO table element.

The event log registers every important event that happens during the operation /such as alarms, operator's interventions etc./ with its date. This log is typed out automatically when such an event accurs. This log is generated by the man-machine communication subsystem /see Section 4.7.1./ and by the data analysis layer /see Chapter 5./.

The plant log registers the value of a set of predetermined variables cyclically /in general, hourly/ but the operator can ask for an extra log at any time /see 4.7.1./. The generation of these logs is carried out by using the so called LOG-SHEET negatives. This negative contains

- the text to be typed out,
- the identity code /GNB, BNB/ of the variables nested by separation marks.

This generation is done by the LOG generator program which fills out the LOG-SHEET negative with the actual values and sends it to a log. A computing background program can be initiated in order to calculate variables for the logging subsystem. An example log is given in *Fig. 20.*

Both the computing and the logging subprograms are initiated by the COML COMLOG scheduler running at the first priority level, which operates on the basis of the IT1T table /see *Fig. 21.*/.

A plant log can be initiated

- at a given date, or cyclically by the timer,
- by the execution of a given group,
- or by the operator /extra log/.

The LOG generator program operates by using the IT1T table which stores every necessary item of data for this subsystem. There is room for 32 elements in the table, the structure of an element is shown in *Fig. 21.*

The post-mortem log stores a time series of data preceeding and following an emergency trip. This log is a cyclic buffer on the disc where the values of a set of variables are stored periodically. The max. length of this buffer is 128 sectors, each sector can store up to 48 variables. It is evident that if e.g. 256 samples are stored in this dedicated area, there is room for only 24 variables.

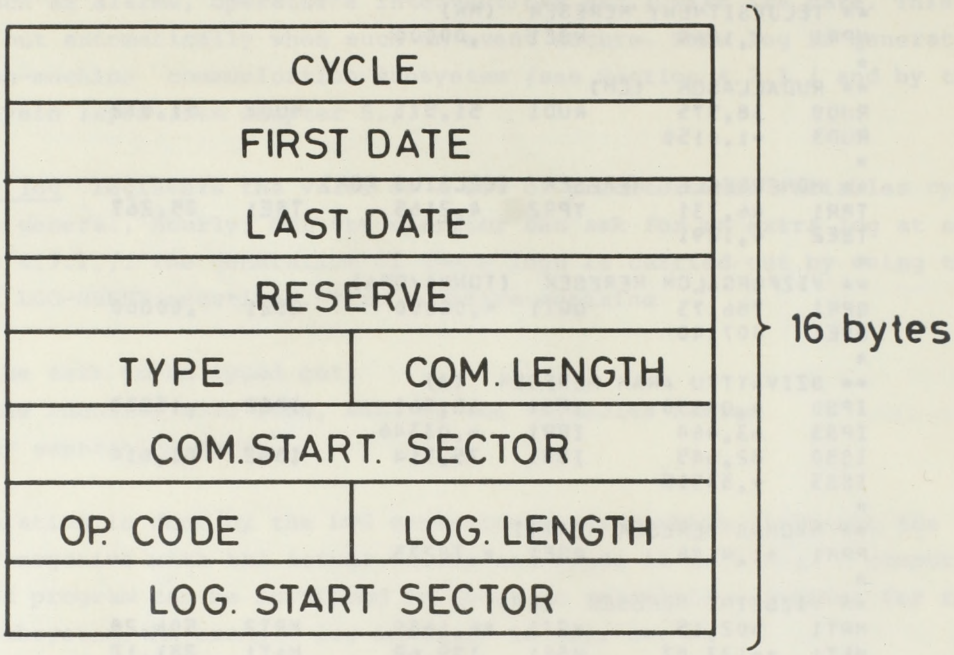
This cyclic buffer is generated by an empty group /having no variables/ with a special primary processing task called PMTASK. This task collects the actual values of the specified variables into the value- and flag-field of the empty group; these fields are subsequently stored on the disc as a post-mortem sample. The PMTASK task is given in Appendix 4.

```
*
****  PROCESS - UZEMI NAPLO  IDOPONT: 27. 14,00  *
*
** TELJESITMENY MERESEK   (MW)
NPR1    4,1064       NSE1    ,00000
*
** RUDALLASOK   (CM)
RUD0    38,575       RUD1    51,571       RUD2    51,072
RUD3    -1,0150
*
** HOMERSEKLET MERESEK   (CELSIUS FOK)
TPR1    46,731       TPR2    4,2165       TSE1    25,267
TSE2    4,1291
*
** VIZFORGALOM MERESEK   (TONNA/ORA)
QPR1    786,73       QGT1    *,00000      QSZ1    ,00000
QSE1    407,40
*
** SZIVATTYU ARAM MERESEK   (A)
IPS0    -,09590      IPS1    65,961       IPS2    ,15383
IPS3    63,464       IPR1    -,03346
ISS0    82,645       ISS1    75,714       ISS2    81,614
ISS3    -,33515
*
** NYOMAS MERESEK   (ATT)
PPR1    *1,4146      PSE1    *,78273
*
** VIZSZINT MERESEK   (CM)
MRT1    502,15       MGT1    *6,1689      MRT2    506,28
MFT1    *-127,87     MSE1    179,62       MAT1    281,12
MAT2    254,90
*
** HUZAT MERESEK   (VOMM)
HAR1    *9,0451      HFR1    *3,5555      HGT1    *33,623
HMK1    6,2726       HFT1    9,1192       HSZ1    *9,1942
*
** RADIOAKTIV AEROSOL KONCENTRACIO   (PICOCURIE/LITER)
RSK1    2,3025       RSK2    1,4440
*
** RADIOAKTIV GAZ KONCENTRACIO   (NANOCURIE/LITER)
RSK3    26,147
*
```

Fig. 20.

Plant log generated by the COMLOG program.

The oldest sample is always overwritten in the cyclic buffer under normal circumstances. When the man-machine subsystem indicates an emergency signal /see Section 4.7.3./, the PMLIST service program is initiated. This permits the overwriting of only half of the cyclic buffer after which the sample collection is inhibited. This solution provides the same amount of data preceeding and following the emergency signal. Using the PMLIST service program the content of the post-mortem log can be dumped either by the line-printer or by the paper tape punch; following this the data collection into the cyclic buffer is again activated. The same service program can also be used for listing the paper tape dumped.



where CYCLE - repetition time (hour, minute)

TYPE - specifies whether this entity is a computing program, or a logsheet, or both

OP. CODE - indicates the logical peripheral demanded

Fig. 21.

IT1T table element.

4.7. Technological operator interface /OPER program/

The task of the technological operator interface is to ensure the communication between the technological operators and the computer. In other words, this interface is the part of the system which gives possibilities for the operators to obtain detailed or general information about the whole actual process in a suitably short time. During operation the operator must have the right in certain cases to alter the parameters belonging to the individual variables. These changes are also made through the technological operator interface. This interface is realized by a program called OPER.

This program can serve four technological operators at the same time. Four operators means four terminal devices independent of each other, which can be displays, typewriters, teletypes, etc. Each of the operators may initiate communication by producing an interrupt i.e. operating a pushbutton or turning a key etc. The four operators do not have equal rights. One of the operators /the so called "chief" operator/ has the sole right to change any parameter whereas the other three /called "assisting" operators/, can only put questions.

We describe the communication between the system and the "chief operator" in Section 4.7.1. In 4.7.2. we point out all differences between the "chief" operator and the assistant operators, and finally we describe the other functions of the OPER program in 4.7.3.

4.7.1. "Chief" operator's communication

The communication between the operator /x/ and the computer is developed in such a way that the operator talking with the system needs to type as few as possible. The program understands the operator's command from the initial letter. These one-letter orders are formed into complete sentences by the system, so the whole written communication is always simple, clear and easy to understand. If the operator produces interrupt the system writes the colon character indicating that the program is waiting for a one-letter command. If for example, the letter "C" is typed the computer completes it to "CHANNEL NAME:". This "C" meant that the operator wanted to choose one variable to ask about, and the program waits for the NAME of the variable. The following letters are accepted by the system after the colon. /The first letters are completed by the program./:

/x/ Operator means always the "chief" operator in this Section.

:ALARM ACCEPTED
:CHANNEL NAME
:EXTRA LOG SHEET
:LIST OF CHANNELS
:NAME OF CHANNEL
:TIME

The variables are designated by their 6-letter MNEMONIC names. If the operator does not know the mnemonic name of a variable he can ask from the system by its identity code. After selecting a variable, the operator may choose which parameter or parameters of the variable he would like to see. For example, let this parameter be the lower alarm limit of the selected variable. The operator pushes down the letter "L", which is completed to "LOWER ALARM LIMIT" and the lower alarm limit is written out. In the same manner the operator can ask the upper alarm limit, last measured value, factors of PID control etc. of the selected variable. Striking down the letter "P" the program lists out the so called primary information of the given variable i.e. measured value, upper alarm limit, lower alarm limit, validity limits, digital control, operating control.

Let us again look at the example. After writing out the lower alarm limit of the given variable the program waits again for a character. Typing the character " . " the system terminates the communication printing out the actual date and time. If the operator wants to alter a parameter, in our example the lower alarm limit, he types the letter "C" /completed to "CHANGE TO"/ and gives the new value of the parameter to the system. *Figure 22.* shows the whole communication of the example.

:CHANNEL NAME: ABCDEF LOWER ALARM LIMIT
123.400 CGRAD CHANGE TO 125.20 .
08.14.30

Fig. 22.

A communication example.

The underlined letters are typed by the operator. Writing out the actual date and time always means that the system accepted the new value /the new lower alarm limit in the example/ and the whole communication /*Fig. 22.*/ appears in the event log at the same time, i.e. change made by the operator can also be read in the event log /see Section 4.6./.

In the case of operator faults /non existent command letter, non existent mnemonic name, wrong new value etc./ the program starts again giving a new " : ". A wrong number may mean a typing error or, in the example, that the new lower alarm limit exceeds the predefined absolute lower alarm limit /see 4.1./.

If a very important parameter or variable is concerned the program may check whether the operator has permission or the right to make the change. The authority is represented to the system by a bit of a digital input. This bit is set by a suitable tool, e.g. pushbutton, key,switch, etc. If permission is given /authority bit = 1/ the operator may alter any parameter. If there is no permission /authority bit = 0/ and the changing of a parameter needs competency checking, then the system, instead of completing the letter "C" to "CHANGE TO", writes out "UNAUTHORIZED ATTEMPT TO INTERVENE" and this message is also duplicated in the event log. The OPER program distinguishes three types of parameters. For changing certain very important parameter types, e.g. scale factors of the linear transformation of the variable, the program always checks the authority bit. For certain less important parameter types /e.g. lower alarm limit, validity limit, etc./ the program looks at the flag byte of the selected variable, where the competency bit /see 4.2./ shows whether the authority bit has to be checked or not. For not very important parameter types /e.g. operating control, digital control/, no checking is made at all.

In addition to parameters of a certain variable the operator can ask for total lists of the variables over limit/not valid, or variables not permitted. The operator may start an extra log as well. /see 4.6./.

Hardware error messages concerning one measuring channel /overflow, invalid measured value/ are sent to the chief operator. The operator has to accept this alarm message. If he forgets it, a warning message /"UNACCEPTED ALARM"/ is generated after each new operator communication.

The operator commands are listed in Appendix 5. where the underlined characters are pushed down by the operator.

It is to be noted that changing the DIGITAL CONTROL of a variable to YES involves the setting of the OPERATING CONTROL to YES, and changing the OPERATING CONTROL to NO involves the setting of the DIGITAL CONTROL to NO /see Section 4.1./.

4.7.2. "Assistant" operator's communication

As mentioned in the introductory part of 4.7. in addition to the "main" operator whose communication possibilities are discussed in 4.7.1. the system can supply another three so called "assistant" operators. These operators can make inquiries about the same things and in the same manner as the "chief" operator. The most important difference between them is that the assistant operators are not allowed to produce any changes. The program - after answering a parameter of a variable asked by an "assistant" operator - stops the communication, writing out the actual date and time whereas in the case of the "chief" operator the program again awaits a character /" . " or "C"/. The other difference is that on the peripherals of the "assistant" operators there is no alarm message, so after " : " the letter "A" is illegal. The last difference is that the extra-log appears on the corresponding assistant operator's peripheral as opposed to the command of the "chief" operator the extra-log is typed out on the listing log peripheral.

4.7.3. Other functions of the OPER program

As described, all four operators signal their wish to communicate with the system by an interrupt. The four interrupts are collected into a 16-bit interrupt-collector. The technological operator interface i.e. the OPER program handles this 16-bit interrupt collector. *Figure 23.* illustrates the bit-pattern of the 16-bit interrupt-collector of the OPER program.

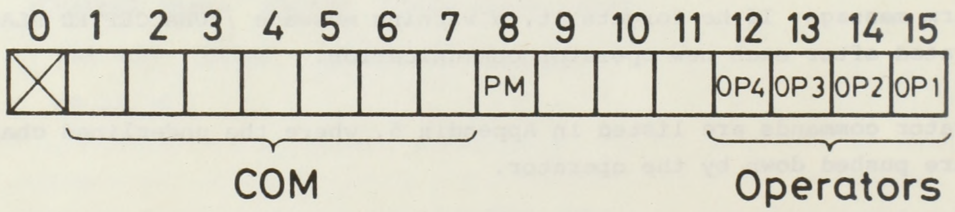


Fig. 23.

Bit pattern of the 16-bit IT collector for the OPER program.

Bits 12-15 represents the IT-s from the operators /bit 15 = chief operator/. An IT coming in at bit 1-7 causes the start of a COM program whose parameter is just the number of the bit, i.e. a number from 1 to 7. This COM program may do seven different things, e.g. counting, listing, etc. according to the IT number.

An interrupt on bit 8 is reserved for the post-mortem log request /see Section 4.6./. Before starting the post-mortem dumping program the system clears the way to it in the background by aborting the running background program - if there is any.

Bits 10 and 11 initiate predetermined messages on the Operator Console. This can serve for the automatic recording of any external event.

CHAPTER 5.

DATA ANALYSIS LAYER

Data analysis means a transformation of the primary data base /gained from the measurements, and described as a disordered set of data by the data acquisition layer/, into a secondary data base having a well ordered form. The principles of ordering - which constitute the basis of the transformation - are the logical connection among the various measured variables. These connections which have to be specified by the user are a priori information about the process and are built into PROCESS-24K's library in advance. These logical relations have to be described in the form of binary logic trees.

The ordering principles built in PROCESS-24K serve the purpose of the alarm analysis. Alarm occurs if the actual value of a measured analogue signal passes a prescribed upper/lower limit, or a digital input indicates an abnormal event. The various logical connections among the alarms /the nodes of the alarm trees/, their possible causes and their consequences form the basis of the transformation of the primary data base. The resulting secondary data base is well-ordered in respect of alarm analysis, and - being the result of a Boolean-type transformation - contains only logical variables. /These are the so-called "deduced alarms"./ Some of the deduced alarms, which have a great importance, are presented to the operator, and may be utilized for reconfiguration of the cyclic tasks /see Section 6.2./.

The duties of the data analysis layer are distributed among three programs running on three different IT levels:

- ANAL is the name of the core resident task having a high /15/ priority, which performs the analysis;
- ALDYS - a lower priority /11/ task carries out the presentation of the alarms;
- ALGEN - a special background program - serves the generation and modifications of the so-called "alarm library", in which the binary "alarm trees" are stored for the ANAL program.

Fig. 24. shows the organization of the information flow among these programs. All three programs have to be replaced if the purpose of the analysis changes.

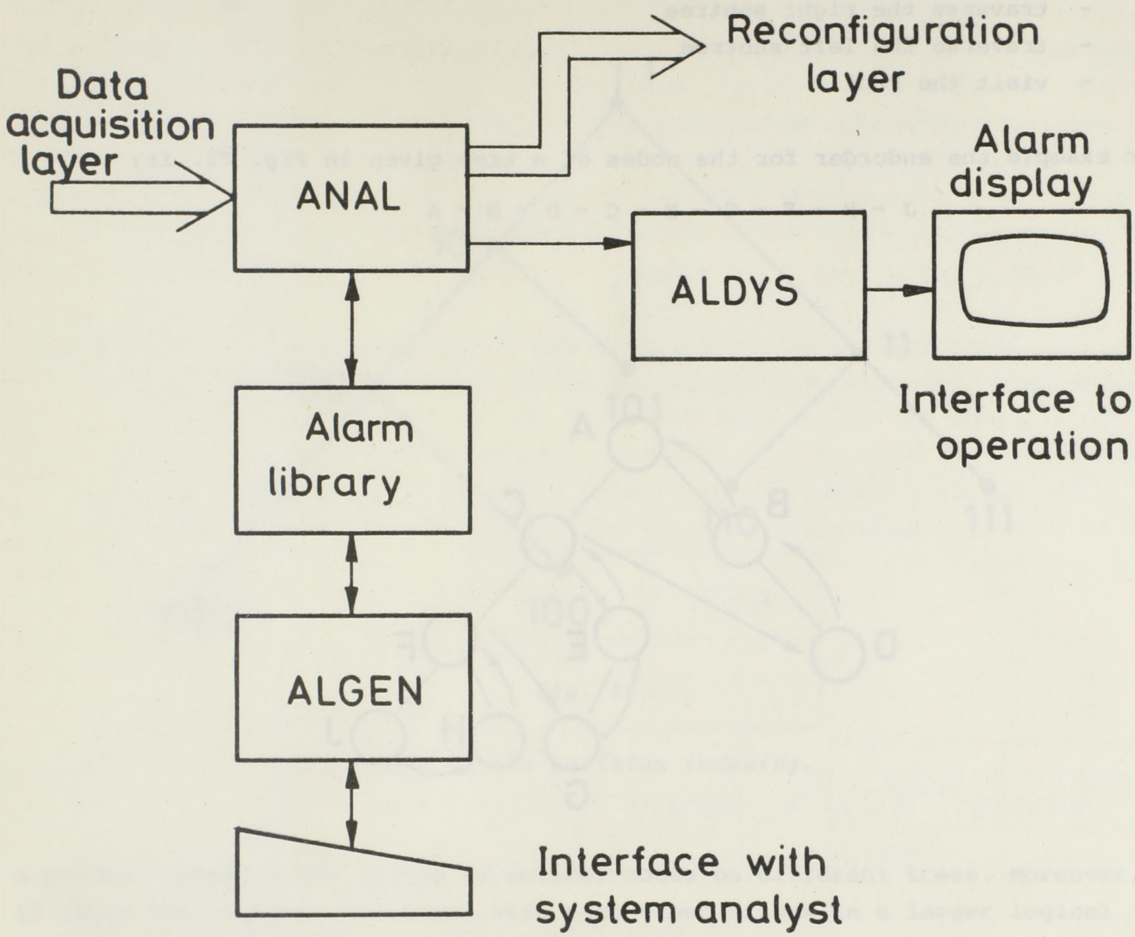


Fig. 24.

Organization of the information flow in the data analysis layer.

5.1. Description of the binary logic trees

The logic trees are described as a list in the library. The terminal nodes of the trees are the elements originating from the primary data base: the alarms, and their logical connections are represented by the nodes on the higher levels. The node on the top of a tree is the root, which represents the result, i.e. the deduction from the analysis. In the course of examination, the nodes of a tree have to be examined systematically so that each node is visited only once. Analysis must start at the terminal nodes and process to the higher levels. The most appropriate method is the so-called End Order Traversal /EOT/ |26|, which contains the following three steps:

- traverse the right subtree
- traverse the left subtree
- visit the root.

For example the endorder for the nodes of a tree given in *Fig. 25.* is:

J - H - F - G - E - C - D - B - A

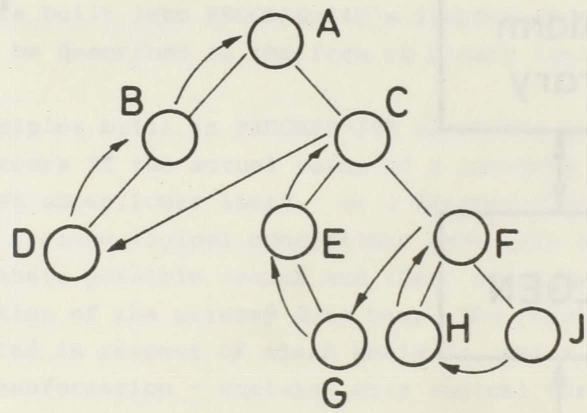


Fig. 25.

Example of endorder traverse.

The position of the nodes on the tree is characterized by two numbers:

- the scanning pointer /SP/ points to the next node in traversal,
- the Node Number /NNB/ is an index which indicates the position and the level of the node on the tree, as shown in an example in *Fig. 26.* /NNB is a binary number whose length shows the level of the node. The last figure of NNB on the left subtree is a 0, on the right subtree is a 1./

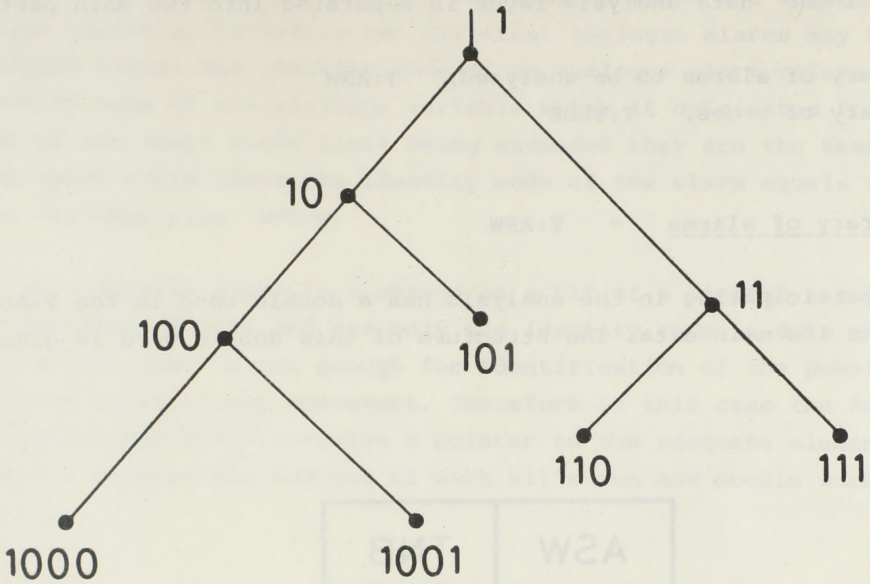


Fig. 26.

Example to the position indexing.

A process variable may belong to several nodes on different trees. Moreover, if there are subtrees identical with each other either in a larger logical connection, or in different trees. It is possible to separate them into independent trees whose roots are applied as terminal nodes on other trees. In this way, the trees may be coupled to each other through their nodes.

5.2. Description of the time relations

The nodes of a given tree - the alarms - represent logical connections among data originating from process variables which

- are measured in different measuring groups, with different cycle time, or
- may change with very different time constants.

Hence, it seemed necessary to assign a relative time to each node of the trees. The time is related to the occurence of the alarm which belongs to the variable, either measured with the shortest cycle time or having the minimum time constant.

5.3. <u>Representation of the Alarm Trees</u>

The library of the data analysis layer is separated into two main parts:

- library of alarms to be analysed, T:ASW
- library of trees, T:TREE

5.3.1. <u>Library of alarms</u> - T:ASW

Each alarm, participating in the analysis has a double word in the T:ASW table which contains its main data. The structure of this double word is given in *Fig. 27.*

<table>
<tr><td>ASW</td><td>TNB</td></tr>
<tr><td colspan="2">IDC</td></tr>
</table>

where ASW - Alarm Status Word

TNB - Tree NumBer - identity number of the first tree which contains the alarm, as a node.

IDC - identity code of the alarm.

Fig. 27.

Structure of the ASW double word.

The alarm according to its origin, may be

- analogue,
- digital,
- deduced alarm.

If the value of a measured analogue variable passes its trip level, an analogue alarm occurs. It is possible to specify two limits: upper and lower, for the same analogue variable, therefore two different analogue alarms may be attached to one analogue signal. The identity code of an analogue alarm refers directly to the identity code of the analogue variable which it originates from; namely in the case of the lower alarm limit being exceeded they are the same, in the case of the upper alarm limit the identity code of the alarm equals the identity code of the variable plus &8000.

The origin of a digital alarm is a change in a bit of a digital input. A digital input contains 16 bits and has only one identity code in data acquisition layer. This single code is not enough for identification of the possible 16 alarms /events/ which it may represent. Therefore in this case the ASW double word is a special one which contains a pointer to the adequate element of a table - T:LOG - storing the address of each bit's own ASW double word. */Fig. 28./*

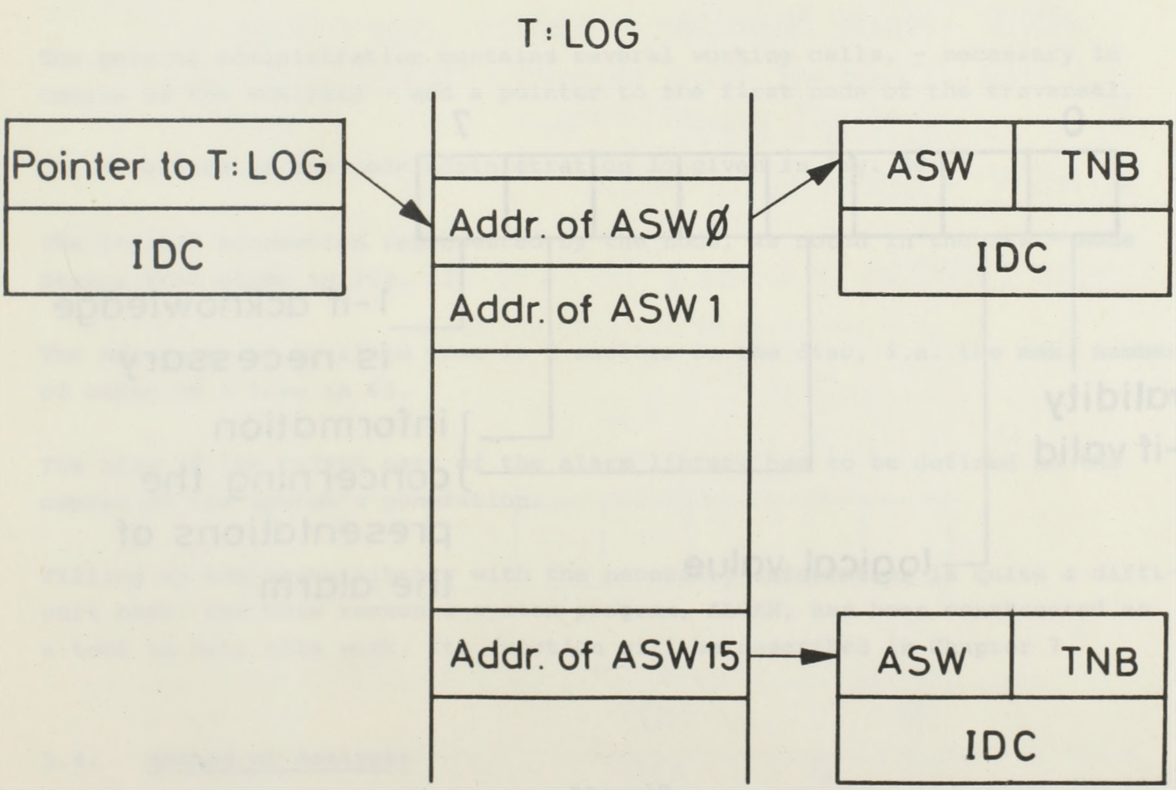


Fig. 28.

ASW double word for digital inputs.

The identity codes /IDC/ belonging to the bits of the digital inputs and to deduced alarms have to be defined in the course of the generation of the alarm library, but &8000 > IDC > &4000 is obligatory for these codes.

The data base defined by the data analysis layer is a logical data base, so it is necessary to define the alarm's logical value. The logical value of an analogue alarm will be equal to one if the analogue variable concerned is in the alarm state.

The logical value of a digital alarm is the value of the bit in the digital input from which it originates. The result of a Boolean equation determines the logical value of a deduced alarm.

The logical value of the alarms is represented in the ASW byte. Within ASW, every bit is interpreted individually and several of them have to be determined in the course of the generation /see Chapter 7./.

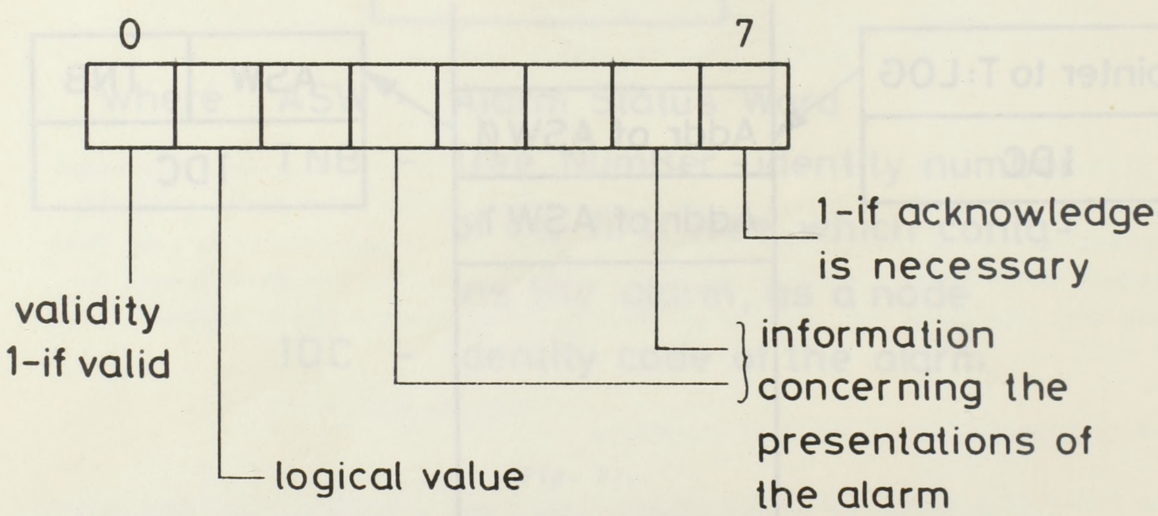


Fig. 29.

Bit pattern of the ASW byte.

The validity bit defines an alarm as not valid, if the value of the variable to which it belongs is not valid, or if its measurement is not operating.

One ASW double word belongs to each alarm even if an alarm takes place on several trees.

The size of T:ASW is 10 sectors on the disc; this is enough room for 640 different alarms. Supposing that an alarm is coupled to two alarm trees - as an average - the size of the T:ASW is enough for 1280 nodes on the alarm trees.

5.3.2. Library of the Binary Tress - T:TREE

The second part of the alarm library contains the description of the trees. Every tree has a general administration /6 words/, and several node administrations /4 words/.

The general administration contains several working cells, - necessary in course of the analysis - and a pointer to the first node of the traversal.

The structure of the node administration is given in *Fig. 30.*

The logical connection represented by the node, is noted in the NSW - Node Status Word given in *Fig. 31.*

The max. size of an alarm tree is 2 sectors on the disc, i.e. the max. number of nodes on a tree in 62.

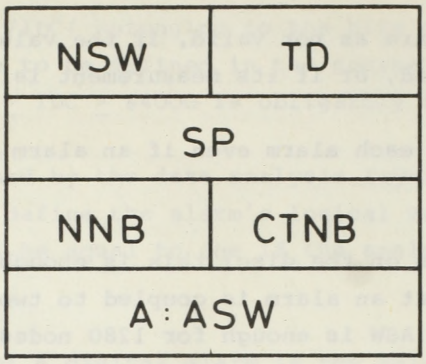
The size of the T:TREE part of the alarm library has to be defined in the course of the system's generation.

Filling up the above library with the necessary information is quite a difficult task. For this reason a system program, ALGEN, has been constructed as a tool to help this work. Its function will be described in Chapter 7.

5.4. Method of Analysis

The analysis is performed by a high priority real time task: ANAL. ANAL may be initiated:

a/ by the data acquisition layer, with a minibuffer, containing the identity code of the alarm, which has to be analysed;
b/ by the timer, without a minibuffer.

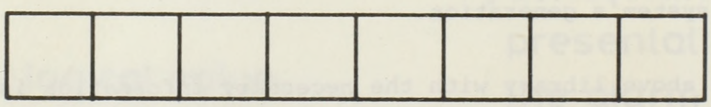


where

NSW - Node Status Word
TD - Time Difference (see 5.2.)
SP - Scanning Pointer (pointer to the next node)
NNB - Node NumBer - position index (see 5.1.)
CTNB - Coupled Tree NumBer - if there is none
CTNB = &FF
A:ASW - Address of ASW

Fig. 30.

Structure of the node administration.



Ø Ø - alarm (input event)

Ø 1 - NOT

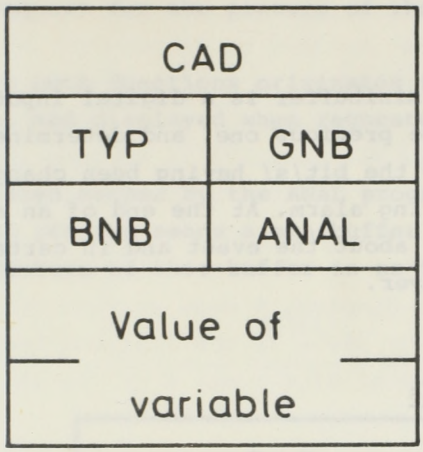
1 Ø - AND

1 1 - OR

Fig. 31.

Bit pattern of Node Status Word.

The structure of the initiating minibuffer is given in *Fig. 32.*:



where

CAD - chaining address

TYP - Type of alarm : upper (lower limit exceeding, digital variable, etc.)

ANAL - if = 1, it has to be analysed
if = Ø, it has to present (send to ALDYS) only.

Fig. 32.

The initiating minibuffer for ANAL.

a/ Tasks of ANAL intiated by a mini are:

1. To evaluate the new logical value of the alarm into the Alarm Status Word.
2. To scan the alarm tree/s/ containing the alarm as a node. If there is a CTNB, different from &FF on the analysed node, the number of the coupled tree has to be written into the Tree Waiting List /TWL/. If the scanned node has a larger TD /Time Difference/ than the TD belonging to the node by which the analysis was initiated, the further scanning has to be delayed. In this case, the identity number of the tree has to be written on the Suspending List. /SL/
3. To analyse the next tree from TWL.

b/ Tasks of ANAL, initiated by the Timer are:

1. To decrement the waiting time on every item of the SL.
2. To continue the analysis if a tree whose waiting time becomes zero occurs in the SL.

If the content of the initiating minibuffer is a digital input variable, ANAL compares its new value to the previous one, and determines the identity code of the alarm, represented by the bit/s/ having been changed. This alarm will be considered, as an initiating alarm. At the end of an analysis the ANAL program informs the operator about the event and in certain cases it initiates the adaptive control layer.

5.5. Presentation of the results

The presentation of alarms in PROCESS-24K is made by the task ALDYS, and appears on two peripherals.

a/ Every alarm is logged - completed with the time of its occurrence - in the event log on the M:LL peripheral.

b/ An alarm display serves for presentation of alarms chosen by the operator. This choice is made by interrupt requests connected to an IT collector of 16 bit length. The bit pattern is the following:

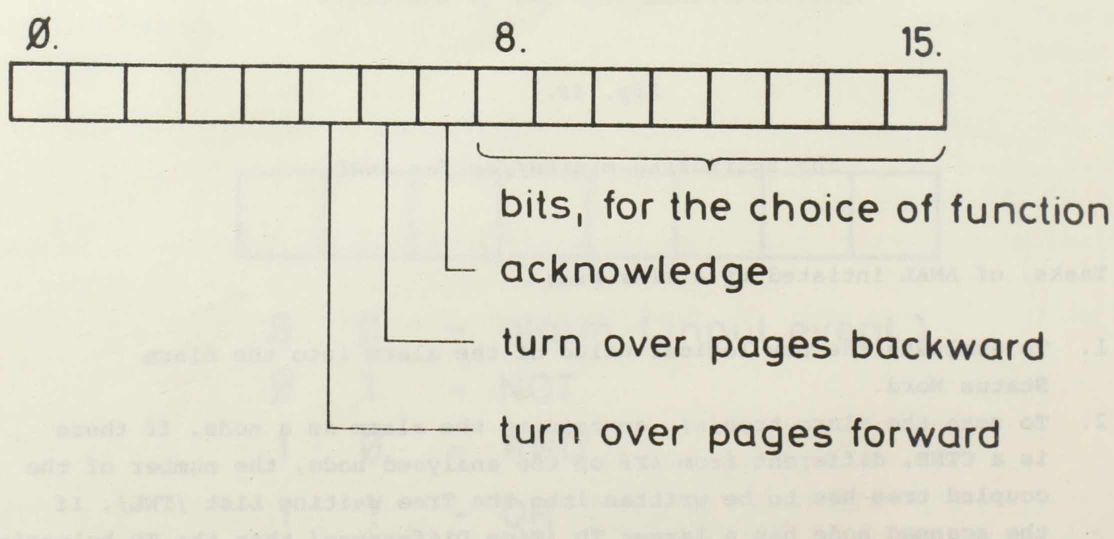


Fig. 33.

Bit pattern of IT collector initiating ALDYS.

8 bits serve for the choice of the function. At present only two of them are used:

- 8 bit: alarm list request
- 9 bit: request for the picture of the alarm trees.

The information for both functions originates from ANAL, and it has to be stored on the disc, and displayed when requested by the operator.

The ALDYS is initiated either by the ANAL program, or by the operator, causing interrupt. The ANAL program sends a minibuffer in the case of each alarm occurence. The structure of this buffer is as follows:

<table>
<tr><td colspan="2">CAD</td></tr>
<tr><td>TYP</td><td>GNB</td></tr>
<tr><td>BNB</td><td>FNK</td></tr>
</table>

where

TYP - type of alarm - as in the mini, initiating ANAL, but completed by one bit:

bit 1 { =0 if the alarm has to acknowledged; =1 if acknowledge is not necessary

GNB, BNB - identity code

FNK = 00 - the information is addressed into alarm list

Fig. 34.

Starting buffer for ALDYS, containing a single alarm.

In this case an alarm message is sent to the event log, and if it has to be acknowledged its identity code will be written into the alarm list on the disc, and into an "acknowledge stack".

The alarm remains on the alarm list as long as it exists. /When an alarm situation is over, a buffer with similar structure is sent from ANAL./ The same remains in acknowledge stack for as long as it remains unacknowledged. The maximum length of this stack is 12 elements. If it becomes full an error message appears in the event log thereby indicating the absence of the operator.

The maximum length of the alarm list is 58 elements, but there is room for only 12 rows on the screen of the display. A page pointer points to the momentary first displayed element of the list. If the list has more than 12 items, it is necessary to page by the use of bits 5 and 6 of the IT collector.

Picture of the Alarm Trees

If at the accomplishment of the analysis of a tree a deduced alarm to be presented is found by ANAL, a mini and a midibuffer are filled in. The content of the mini is given in *Fig. 35.*

The midibuffer contains the identity codes of the alarms on different nodes, in order of sequence. The first in order has the Node NumBer /NNB/ equal to 1, hence this alarm is on the root. The alarms are enumerated in order of increasing NNB.

Pictures have to be stored on the disc, and displayed only if the 9th bit of the IT collector is activated. Hence there is a "Picture List" on the disc where the PNB of pictures and a disc address are stored. The disc address assigns an area with 32 words length, where the whole content of midi /sent by ANAL/ is stored.

The maximum number of elements of the picture list is 60. The selection of the required picture is possible through bits 5 and 6 of the IT collector. The alarm tree, which may be displayed on the screen, has a maximum of 15 nodes /4 levels/, and this fits the size of the midibuffers too. In the case of a larger tree, the picture description is sent in a chain of midis by the ANAL, and is stored on chained territories on the disc.

<table>
<tr><td colspan="2">CAD</td></tr>
<tr><td>TYP</td><td>—</td></tr>
<tr><td>PNB</td><td>FNK</td></tr>
<tr><td colspan="2">Pointer to
midi</td></tr>
</table>

where TYP - type of the picture
&40 - new picture
&20 - already presented, but it has to modify
00 - erase, the picture has to be cancelled

PNB - picture identity code

FNK - &01 - presentation of tree

Fig. 35.

Starting buffer for ALDYS in the case of tree presentation.

CHAPTER 6.

ADAPTIVE CONTROL LAYER

In the life of a plant there are different phases where the conditions of the operation and the goal of the control are quite different. For example during a start-up period the goal of the control is to reach a predetermined working point according to a given trajectory while during normal operation the goal is to stabilize the operation in the vicinity of this working point. Moreover in an emergency state quite different variables are important from those in a normal operating condition. For this reason if a control system has to operate in a wide range of states of a plant, there must be a reconfiguration layer which adapts the control system to the changing conditions.

In the PROCESS-24K system this adaptation can be initiated either

- by interruptions of the outer world, or
- by software means if e.g. the data analysis layer indicates an abnormal status.

These two initiations are not identical. By hardware interruptions the actually operating groups remain working further but new groups also measure and control in the system. This extra measurement and/or control happens only once, it is like a snapshot of the plant.

On the other hand by software means the cyclic measurement and/or control tasks can be rearranged, in this way reconfiguration of the complete real-time system is done.

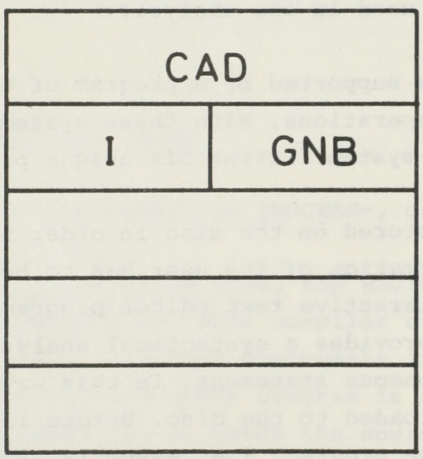
6.1. <u>Troubleshot measurements</u>

Hardware interruption can initiate single measuring and/or control groups. These interrupt requests are handled by an IT collector card /74.880/ which initiates the HWIT program. This program distinguishes 8 different interrupts /Bits 0-7 of the interrupt collector/ and starts the measurement of the group belonging to the given interruption as if it were the timer. The assignment of a group to a given interrupt is stored in the THWIT table. This table is 8 bytes long, the byte 0 corresponds to interrupt 0, etc., the content of the byte is the Group NumBer /GNB/ of the group. If there is no group connected to an interruption, the content of the corresponding byte is &80. In order to initiate the measurements very rapidly, the HWIT program works at level 21 and its initiation has priority over the cyclic starts in the measurement subsystem.

6.2. Reconfiguration of the cyclic tasks

In different operating states of a plant, different measurement/control actions are needed. For this reason the HWIT program can change the periodic real-time tasks of the system. The timer initiates the measuring subsystem by using a TCIK table and the HWIT program can overwrite this table. Every element of this table corresponds to a given cycle time and the content of a given element is the group number /GNB/ of the first measuring group with this cycle time.

When the HWIT program is started by another program /i.e. alarm analysis program/, the M:STRT monitor module is used with the initiating minibuffer given in *Fig. 36*.



where CAD - chaining address
I - correponds to i in the $2^i T$ cycle time equation ($0 \div 7$)
GNB - group number of the new group

Fig. 36.

Initiating minibuffer for the reconfiguration.

CHAPTER 7.

SYSTEM GENERATION

As a general process control system PROCESS-24K is empty, if somebody wishes to use it, he has to specify

- the real-time environment,
- the measuring groups and their respective cycle times,
- the real-time tasks,
- the automatically displayed variables,
- the required log-sheets,
- the alarm trees used in the analysis.

Each of these problems is supported by a program of the system library and here we summarize their operations. With these system programs the user can generate his own PROCESS system fitting his unique problem.

Every source program is stored on the disc in order to save time. For this reason every source information of the user has to be loaded into the disc initially. The /TEXTE interactive text editor program loads the disc, but before loading, it also provides a syntactical analysis of the information and it rejects every erroneous statement. In this way only syntactically correct programs can be loaded to the disc. Before loading the user has to specify the type of source program, i.e. assembly, PROCESS language, log-sheet negative or generation information. This checking increases considerably the processing speed of the background system generator processors.

7.1. Description of the real-time environment

Before loading real-time tasks into the system one has to specify the symbolic names and the physical addresses of the input/output devices of the real-time measuring subsystem. This information is stored on the disc in the RTDATA file and, for example, the PROCESS compiler uses this data during the compilation. In the system programs for naming real-time peripherals only those symbols can be used which were defined in this phase and are stored in the RTDATA file.

As we have seen in Section 2.2. the structure of the disc depends also on the number of groups used. For this reason in this defining phase the user has to specify

- the symbolic names of the groups
- either the cycle time or the initiating interrupt request of each group
- the post-mortem data /number of samples, cycle time/.

This information determines the structure of the disc moreover the content of the different control tables /T:CIK, T:HWIT/. For this reason this data is stored on the disc in the GRDATA file and used during the starting phase of the actual PROCESS system for evaluating different system constants, furthermore, the use of this information results in the generation of the structure of the disc. After specifying the RTDATA information one has a system with an empty user library with no real-time tasks, but the timing of the system already meets the requirements of the user and the disc is formed for the later operation. This system specification is carried out by the $GENES program.

7.2. Loading of the real-time tasks

Real-time tasks can be written either in PROCESS-, or in assembly language.

If the PROCESS high level language is used, the ×AUTOC compiler generates the object code from the source program. This compiler can generate a list of the compilation by indicating the erroneous statements in order to facilitate the error analysis. The loading of a correct program is done by the @LINK loader. This loader calls the compiler, so it needs the source program and the name of the group where the given program has to be loaded. In the case of assembly programming the &BSATR assembler produces the object code from the source program. The object code is stored temporarily in the user library and the assembly task is loaded from here to the wanted group by the .TASLK loader. With this loader one can also delete or replace a given task. In addition, external functions can also be loaded. All these loadings are carried out in a working system, so the system can be built or altered during its operation. When a real-time program is loaded, it is in an inactive state so it does not operate, therefore the technological operator has to activate every real-time task.

7.3. Loading of the COMLOG programs

The COMLOG programs /see Section 3.3./ form the COMLOG library. This library consists of two types of files: computing programs and log-sheet negatives. The computing program written in assembly language is compiled by the $BSATR assembler and is stored temporarily in the user library. From here the object program is loaded into the COMLOG library by the +BSALD loader.

The source log-sheet is checked by the 'LOGTR compiler which gives a compilation list and if a fault occurs, indicates the number of the erroneous line and the type of the error. The "LOGNL loader loads the correct log-sheet negative into the COMLOG library. This loader calls the 'LOGTR compiler and indicates to it that the generated log-sheet negative has to be loaded into the COMLOG library.

Apart from loading, +BSALD and "LOGNL can also replace the old file in the COMLOG library by a new one.

In order to link the computing program and the logsheet with each other and with the starting conditions the =COMLK linking program fills in the IT1T table, which controls the background activity of the system. With this program the IT1T table can be modified, i.e. the computing program and the log-sheet can be replaced, the starting conditions can be changed.

7.4. Specification of the post-mortem log

In order to generate a post-mortem log, the user has to specify

- the number of samples and the timing of the cyclic buffer of this log,
- the actual variables chosen for this log.

The former is done during the specification of the real-time environment because it has consequences affecting the structure of the disc. For this reason an empty group has to be defined in the GRDATA information and a post-mortem disc area must be linked to this group with the necessary sampling rate.

The variables of the cyclic buffer are collected by a PMTASK real-time task connected to the defined empty group. This task contains a table which stores the identity codes of the variables selected for the post-mortem log. The user has to fill in this table in the source program of the PMTASK /see Appendix 4./ with the identity codes of the variables. It is possible to name the variables with their mnemonics instead of the identity codes, using the DATA,S directive of the &BSATR assembler. When this program is ready, it has to be linked to the empty post-mortem group by the .TASLK loader /see Section 7.2./.

7.5. Specification of the data presentation

The automatic data display function of the system is controlled by the TDIGO table /see Section 4.5./ and there is a system program, TDIGO, by which the user can modify the content of this table. This program enables the user

- to request a list of the TDIGO table,
- to change its content.

To specify the TDIGO table the user has to know the mnemonics of the variables to be displayed and the symbolic names of the desired outputs. The TDIGO program provides conversion into physical addresses. In order to reduce the used CPU time, the user must feed the information into the system by a paper tape. The tape is checked while it is being read, and if an error occurs, the program indicates error on the Console, where the user can correct on-line the erroneous statement. The program uses the M:EI peripheral as input and the M:LO peripheral prints out the lists.

7.6. Loading of the alarm library

ALGEN /Alarm Generator/ enables the user to load the alarm library with the necessary data, in the course of a conversation on the Operator Console.

The functions of the program are the following:

- to initiate the library, i.e. to perform the necessary clearing of the library's territory on the disc, before the first loading;
- to fill in the T:LOG and T:ASW tables by asking the identity codes of the different alarms/events belonging to each bit of the digital variables;
- to assemble the data of the alarm trees, and to load them into the T:TREE area of the library;
- to give an opportunity for the deletion or the modification of the digital variables and the alarm trees, moreover, for the modification of several nodes of a given tree;
- to handle the catalogue of the alarm library, and - according to the user's command - to print out its content on the M:LO peripheral.

In the case of nodes belonging to several trees, it is the program's duty to produce /or delete/ the necessary couplings among the nodes of different trees.

Informations about the presentation of digital and deduced alarms, the need for operator's acknowledgement and the delay times defined to the nodes, will be requested from the user and loaded into the library as well.

REFERENCES

1. P.Elter, R.Roessler: Real-time languages and operating systems. Preprints of IFAC Conf. on Digital Computer Applications to Process Control. 1977. pp. 1-12. The Hague. North-Holland Publ.Co.

2. Instrument Society of America: Industrial Computer System. FORTRAN 1972.

3. Purdue Europe Real-Time Basic Committee: Report on Real-Time Extensions and Implementation of Real-Time BASIC. First report. 1975.

4. Digital Equipment Corporation: INDAC-8 Software

5. P.Hugot, M.Ritout: PROCOL: un nouveau language de programation temps réel sur MITRA 15. Automatisme pp. 290-296. No. 5. Mai. 1974.

6. RTL MITRA. Manual de référence. 1976. Compagnie International pour l'Informatique No. 5569/u/FR

7. K.H.Timmesfeld et el.: PEARL - a proposal for a process - and experiment automation real-time language. 1973. Karlsruhe. Report KFK-PDV 1.

8. P.M.Woodward, P.R.Wetherall, B.Gorman: Official Definition of CORAL 66. 1973. HMSO publication SBN 11 470221-7.

9. BICEPS - Basic Industrial Control Engineering System. GE Software for the THAC 4000-1972.

10. A.D.Heher: Some features of a real-time BASIC executive. Software Practice and Experience. No.6. 1976.

11. R.Green, N.Estcourt: The use of high level languages in real-time systems. Proc. of the Conference on "Software for Control" pp.68-74. 1973. IEE publ. No. 102., London.

12. T.B.Mahood: Computer control of the Bruce Nuclear Power Station. Proc. of the 2nd Conference on "Trends in on-line Computer control systems". pp. 81-92. 1975. IEE publ. No. 127, London.

13. T.E.Dy Liacco: Digital computer applications in the control of electric power systems. Preprints of the IFAC Conf. on "Digital Computer Applications to Process Control". pp. 733-741. 1977. The Hague. North-Holland Publ.Co.

14. N.T.C. McAffer: The computer instrumentation of the Prototype Fast Reactor. Seminar on the application of on-line computers. OECD ENEA OLC/14. pp. 351-379. Norway. 1968.

15. R.Anderson, J.A.Robertson: System software for the control and protection programs of a nuclear power reactor. Proc. of the International Computing Symposium. pp. 145-151. 1975. North-Holland Publ. Co.

16. Kovács B.K.: Az R-10 számitógép PROCESS-8K programrendszere /PROCESS-8K system for the R-10 computer./ Információ Elektronika 1975/3.

17. B.K. Kovács: Ein programsystem zur Messwerterfassung und Prozess-Steuerung für den Digitalrechner ES 1010. VIDEOTON Software Nachrichten 1975/2.

18. PROCESS-16K mérésadatgyüjtő rendszer felhasználói kézikönyv. /User manual for the PROCESS-16K data acquisition system/ VIDEOTON Fejlesztési Intézet, Budapest, 1975.

19. Process Control Monitor. User Manual. VIDEOTON VT-201.085.10.01-SW. 1975.

20. J. Biri, J. Lukács, L. Somlai, Gy. Vashegyi: Industrial data logging system. 1st International Symposium on CAMAC. Luxembourg, 1974.

21. P. Vernet, M. Lagier, R. Husson: A new architecture for process control computers: NARCIS project. Euromicro. 1976. pp. 34-39.

22. E.Csörnyei: Egyszerü overlay és swapping technika R-10-re. /A simple overlay and swapping technique for R-10/ VIDEOTON Software Tájékoztató 1975/3-4.

23. E. Csörnyei: Einfache Overlay- und Swapping-Technik für ES 1010. VIDEOTON Software Nachrichten 1976/1.

24. ES-1010 Reference Handbook VIDEOTON VT-201.017.11.01.SW. 1974.

25. MITRA-15. Manual de présentation. 1972.

26. D.E.Knuth: The Art of Computer Programming.
Addison-Wesley. Amsterdam. 1968.

27. L. Bürger, E. Zobor: On-line alarm analysis of a WWR-SM research reactor.
To be presented at the International Symposium on Nuclear Power Plant Control and Instrumentation.
Cannes, 1978.

28. PROCESS-8K. User Manual.
VIDEOTON VT-208.020.10.01-SW. 1975.

29. PROCESS-8K. Operator guide.
VIDEOTON VT-208.021.10.01-SW.1975.

Appendix 1.

ADDRESSES IN THE R-10 CONFIGURATION

Interrupt system

IT level	DVA word	Status address
1	6802	-
2	5003	0231
3	6402	-
4	6202	-
5	6004	-
6	6008	-
7	6100	-
8	6011	-
10	6200	-
11	4103	0431
13	6080	-
14	6040	-
15	6102	-
16	6009	-
17	6082	-
18	4803	0031
19	6042	-
21	4023	0631
24	6403	-
26	6005	-
29	6010	-

Peripheral system

Interface	Input address	Output address
Display	1C04 and 0024	1E04
Typewriters	002C	000C and 001C
Tape reader	0008	0008
Tape punch	0018	0018
RT clock	0023	-
RT coupler	1E32	-

Appendix 2.

MONITOR MODULES OF THE OPERATING SYSTEM

TRAP	-	aborts a user program when an error occurs and types out the cause of the fault
EXIT	-	instructs the executive that the issuing program has completed its execution
TASK	-	instructs the executive to start a given task
HXBN	-	converts a hexadecimal character string into a binary number
BNHX	-	converts a binary number into a 4 character long hexadecimal string
DCBN	-	converts a decimal character string into a binary number
BNDC	-	converts a binary number into a 6 character long decimal string
DCFL	-	converts a decimal character string into a floating point number
FLDC	-	converts a floating point number into an 8 character long decimal string
CMPA	-	compares two variables
CMPS	-	compares two byte strings
BIBL	-	looks for a given program in the library
LOAD	-	loads a given program into the swapping area
STRT	-	initiates an interrupt program with loading parameters
ABIO	-	aborts every I/O activity of the issuing IT level and frees the buffers used
CLS	-	performs a section call in an overlayed program
RTS	-	returns to the calling section in an overlayed program
FLAG	-	reads/writes a given flag in the semaphor field of the monitor
ADRS	-	reads a given entity from the address field of the monitor
NAME	-	reads/writes a given entity in the name table of the system
KEY	-	displays information on the front panel of the CPU
DATE	-	provides the actual time

GET	-	provides the actual value and the flag of a given variable
PUT	-	gives a new value to a given variable
PUTF	-	gives a new flag to a given variable
96O	-	writes out a digital output
MINI	-	reserves a mini buffer /10 bytes/
MIDI	-	reserves a midi buffer /32 bytes/
MAXI	-	reserves a maxi buffer /256 bytes/
FREE	-	frees a buffer
ZIO	-	performs an input/output action
ZDIO	-	performs a disc transfer
ZWAT	-	waits for the execution of a determined I/O action
ZTYP	-	types out a message and subsequently waits for a reply.

Appendix 3.

SUBROUTINES IN THE PRIMARY DATA PROCESSING

1. Fixed format instructions

SCALE	-	linear transformation /AX+B/
LINn	-	performs linearization, where n is a parameter between 1 and 7. Three linearizations are built in for thermocouples of different types, the remaining 4 are reserves
CONV	-	calls a conversion subroutine written in free format
VALIA	-	validity checking against two absolute limits
VALID	-	validity checking against changing speed
LIMU	-	comparison with a lower limit
LIMUL	-	comparison with a range
FILTn	-	performs an n-th degree exponential filtering /$0 \leq n \leq 7$/ and stores the value in the data base

2. Free format instructions

Arithmetic instructions

The parameter of these instructions is either the identity code of a variable, or a constant

SUM	-	addition
EXTR	-	subtraction
MUL	-	multiplication
DIV	-	division
POWER	-	evaluates the X^n function

Logical instructions

These instructions are always referred to the bits of the A accumulator, the result remains in the A accumulator

MASK	-	masks the accumulator with a constant or a variable
NAND	-	logical NAND operation between 2 bits
NOR	-	logical NOR operation between 2 bits
EXOR	-	exclusive OR between the accumulator and a variable
ROTn	-	rotates cyclically n steps the A accumulator to the right

LDO	-	loads zero into two bits
LD1	-	loads one into two bits
LDB	-	loads one bit from a variable
LDBN	-	loads the negation of one bit from a variable
STB	-	stores one bit into a variable
STBN	-	stores the negation of one bit into a variable
CHB	-	changes two bits with each other
CHBN	-	complements two bits, afterwards it changes them
GPK	-	calculates the number of ones on the bit positions determined by a mask and if this is greater than or equal to a c constant $/0 \leq c \leq 15/$, it stores 1 in the j-th bit position of the accumulator $/0 \leq j \leq 15/$, otherwise it stores zero in this position.

Data moving instructions

SET	-	loads a variable or a constant into the A, E accumulators
SETUP	-	loads the upper limit of a variable into the E, A accumulators
SETLO	-	loads the lower limit of a variable into the E, A accumulators
STORE	-	stores the constant of the A, E accumulators or a constant into a variable
STT	-	storing a bit serial starting with the i-th bit in the A accumulator into a variable starting with the j-th bit
STTN	-	similar to the STT operation but first it complements the bits before storing.

Branching instructions

BRU	-	jump without condition
BRUI	-	indirect jump
IFAC	-	jumps if the content of the E accumulator is negative
IFVAL	-	jumps if the variable is invalid
IFLIM	-	jumps if the variable is in overlimit state
IFLIML	-	jumps if the variable passes its lower limit
BRUO	-	jumps if 2 predetermined bits in the A accumulator are zeros
BRU1	-	jumps if 2 predetermined bits in the A accumulator are ones
CYCLE	-	jumps if a looping condition is fulfilled

Control instructions

LIMI	–	limits a variable between 2 values
ON	–	activating a channel
OFF	–	inactivating a channel
BIT	–	transfer of digital output
DA60	–	transfer of analogue output through D/A converter
BP	–	time modulated digital output
PID	–	calculates a variable according to a prescribed PID algorithm

Function generators

These instructions are always referred to the E, A accumulator and the result also remains there.

SQUARE	–	generates the 2nd power of the variable / $y = x^2$ /
SQRT	–	square root extension / $y = \sqrt{x}$ /
RECIPR	–	generates the reciprocal value /$y = 1/x$ /
INT	–	generates the integer part of the variable
ABS	–	provides the absolute value of the variable
EXP	–	exponential function / $y = e^x$ /
RANGE	–	regards the less significant 4 bits of the A accumulator as a binary number / $0 \leq n \leq 15$ / and generates the / $\sqrt{10}$ /^{n} function
FLOAT	–	converts the content of the A accumulator into floating point representation
EXFNCn	–	external function calling /n = 0 – 4 /. The user can freely define up to 5 function generators in every group.

Program transfer instructions

FIN	–	finishes the execution of a block and the processing begins to work on the next channel
CCAD	–	transfers the execution to the chained next block, otherwise the execution of the next channel is started
RET	–	return from subroutine.

Appendix 4.

PMTASK POST-MORTEM LOG GENERATOR TASK

```
                   * PMTASK    1977.09.06.
                   *
                          CDS
          0006 ZC         EQU     6
          0020 CHAIN      EQU     &20
          0034 HAROME     EQU     &34
                          FIN
                   *
                   L      LDS
     0000 8024            DATA    &8000+START
     0002                 RES     4
     000A 50              TEXT    "PMLIST"
     000B 4D
     000C 4C
     000D 49
     000E 53
     000F 54
                   *
                   *   LIST OF THE DESIRED VARIABLES:
                   *
     0010 0301     VTK    DATA,S  NPR1
     0012 0300            DATA,S  PPR1
     0014 0302            DATA,S  QPR1
     0016 0303            DATA,S  TPR1
     0018 FFFF     VTV    DATA    &FFFF                 * END OF THE LIST
                   *
                   *  PROGRAM:
                   *
     001A 0010     VTAB   DATA    VTK
     001C 0000     I      DATA    0                     * GNB,BNB
     001E          K      RES     1
                   *
     0020          X      RES     1
     0022          FLAG   RES     1
                   *
                          FIN
          0001            LPS     L
     0024 F104     START  XAX
     0026 151C            SBR     I                     * GNB
     0028 4220            LDX     #CHAIN
     002A F205            ICX     =5
     002C 8E06            LBR     @#ZC,X                * PMPR
     002E 2DFF            LBL     =&FF
     0030 F11C            CNA
     0032 151D            SBR     I+1
     0034 111E            STA     K
     0036 2207            LDX     =7
     0038 8E34            LBR     @#HAROME,X            * PM COUNTER
     003A 2DFF            LBL     =&FF
     003C 171C            ADM     I
     003E 0410            LEA     VTK
     0040 111A            STA     VTAB
     0042 2200            LDX     =0
     0044 1320     CIKL   STX     X
```

```
0046 A21A         LDX     @VTAB,X
0048 F905         LDR     #5
004A 0B18         CMP     VTV
004C C00D         BE      EXIT
004E F70E         CSV     #&0E        * M:GET
0050 1322         STX     FLAG
0052 021C         LDX     I
0054 F70D         CSV     #&0D        * M:PUT
0056 021C         LDX     I
0058 0022         LDA     FLAG
005A F712         CSV     #&12        * M:PUTF
005C 001E         LDA     K
005E 171C         ADM     I
0060 0220         LDX     X
0062 F202         ICX     #2
0064 CF10         BRU     CIKL
             *
0066 F701    EXIT CSV     M:EXIT
FFFC 0000         FIN
FFFE 0024
FF00 0101         END
FF02 0068
FF04 0000
```

Appendix 5.

OPERATOR COMMANDS

1. :ALARM ACCEPTED
DD, HH,MM

2. :EXTRA LOG-SHEET
COMLOG NAME: XXXXXX
DD, HH,MM

3. :TIME
DD, HH,MM

4. :LIST OF CHANNELS NOT PERMITTED:
DD, HH,MM

5. :LIST OF CHANNELS OVER LIMIT/NOT VALID:
DD, HH,MM

6. :CHANNEL NAME: XXXXXX WORK LIST:
DD, HH,MM

7. :CHANNEL NAME: XXXXXX PRIMARY
INFORMATIONS:
MEASURED VALUE :
UPPER LIMIT :
LOWER LIMIT :
VALIDITY LIMITS :
DIGITAL CONTROL :
OPERATING :
DD, HH,MM

8. :CHANNEL NAME: XXXXXX MEASURED VALUE:
DD, HH,MM

9. :CHANNEL NAME: XXXXXX CHANNEL NUMBER:
DD, HH,MM

10. :CHANNEL NAME: XXXXXX OPERATING:
DD, HH,MM

11. :CHANNEL NAME: XXXXXX DIGITAL CONTROL:
DD, HH,MM

12. :CHANNEL NAME: XXXXXX TAKE NEW VALUE
AND USE AS VALID :
DD, HH,MM

13. :CHANNEL NAME: XXXXXX SET-POINT:
DD, HH,MM

14. :CHANNEL NAME: XXXXXX FACTORS OF PID
CONTROL:
DD, HH,MM

15. :CHANNEL NAME: XXXXXX FACTORS OF SCALE:
DD, HH,MM

16. :CHANNEL NAME: XXXXXX LOWER ALARM LIMIT:
DD, HH,MM

17. :CHANNEL NAME: XXXXXX UPPER ALARM LIMIT:
DD, HH,MM

18. :CHANNEL NAME: XXXXXX VALIDITY LIMITS:
DD, HH,MM

19. :CHANNEL NAME: XXXXXX BIT NO: XX
DD, HH, MM

20. :CHANNEL NAME: XXXXXX BIT NO: 16
DD, HH,MM

62.529

Kiadja a Központi Fizikai Kutató Intézet
Felelős kiadó: Gyimesi Z.
Szakmai lektor: Keresztély Sándor
Nyelvi lektor: Harvey Shenker
Példányszám: 250 Törzsszám: 78-183
Készült a KFKI sokszorositó üzemében
Budapest, 1978. február hó