Control engineering programme package

Oosterbaan, A.M.

Award date:
1975

1936 bse

AFDELING DER ELEKTROTECHNIEK
TECHNISCHE HOGESCHOOL
EINDHOVEN
Groep Meten en Regelen

CONTROL ENGINEERING PROGRAMMING PACKAGE

door A.M. Oosterbaan

Rapport van het afstudeerwerk
uitgevoerd van mei 1974 tot mei 1975
in opdracht van prof. ir. F.J. Kylstra
onder leiding van ir. J.J.H. v. Nunen

304694

CONTENTS

Foreword

The digital computer with its supporting software languages and peripheral input-output devices is essentially a new tool extending the capacities and resources of the human mind.

However, the access to these tools presents a large number of difficulties to the potential user.
The main difficulties can be enumerated as follows -

a) Language
The nescessity of mastering a computer language such as FORTRAN , COBOL or ALGOL .

b) Algorithms
Sufficient knowledge of algorithms and their implementation.

c) Man/Machine Interface
A working knowledge of the installation dependent input,output and storage facilities is required.

The aim of this project will be to remove as far as possible the above mentioned difficulties , so that the potential user will have a more immediate access to the potentialities offered by the digital computing machine .

1.00 Introduction

1.01 The On-line Design of Control Systems

" Almost all design procedures are an iterative process during which the designer continually makes decisions based on the results of calculations and his experience in the design art" N. Munro (Ref. 14). It is highly desirable that the designer is able to concentrate his thoughts on the correct sequence of decisions rather than that he is pre-occupied with tedious numerical computations. The digital computing machine has the capacity to solve numerical problems with great speed and accuracy. What is needed however is an efficient and effective scheme to exploit this potentiality.

The science of control engineering does have a large number of powerful design tools such as the Nyquist, Bode and the Rootlocus diagrams. However, they involve rather tiresome calculations except in the most trivial cases. The designer would spend more time and energy on actually making these diagrams than he would on the analysis and interpretation . Frequently used and routine procedures such as Nyquist diagrams should be effortless-ly accessible to the potential designer. To put it simply " The designer should be able to communicate with the computer in an efficient manner" .

In general this is not the case. The path to the computing machinery as a tool , is paved with all kinds of difficulties - computer languages such as Algol 60 or Fortran, numeric input problems and program idiosyncrasies of already existing programs. Depending on the design problem the designer quickly reaches the "break-even point " in terms of time and energy to be invested, and will revert to the old paper and pencil methode.

The aim of this project will be to enable the potential designer to communicate with the computer in an effective and efficient manner. A system of meta-programs must be designed so that all existing computer programs and future programs will be directly accessible to the designer by simple commands on a teletype keyboard or by commands on punched cards for a cardreader.

1.02

The Aim of the Control Engineering Program Package

The aim of the Control Engineering Program Package will be to design a conversational on-line facility for designers in control engineering. By on-line is meant that the response time of the communication between the designer and the computer is within reasonable limits. By conversational is meant that the designer can communicate with the computer via a teletype or via a visual display.

The reasons for desiring such a facility can be enumerated as follows-

a) All control engineering techniques such as Nyquist, Bode, Rootlocus etc. will become directly accessible to the potential designer.

b) The input information to computer programs is greatly facilitated. Possible errors are noticed immediately and can be corrected with minimum of effort.

c) The form of the output information can be determined by the designer as the analysis of the problem progresses. At one stage of the design process the designer might desire only numeric output via the teletype whereas later he might demand a hardcopy from a plotter device.

d) The numeric value of variables can be altered as desired.

e) The designer can follow through a sequence of control programs. Each program using the output from the previous program as input.

The end result will be to realize an optimal interface between man and machine. The computer doing all the computational and routine work and the man doing the thinking. At the present this is not so. First the user must familiarize himself with a particular computer language and the particular type of control instructions before he can start with his problem. Sometimes programs for the solution of the problem already exist but are hardly accessible without time and effort.

1.02

One of the most important aims of the Control Engineering Program Package is to make all programs acessible to potential designers without any knowledge what so ever of computer languages or of computer system organization. A user should not be encumbered with anything except the problem he wishes to solve and then as effectively and efficiently as possible.

According to Mr. Munro (ref. 14) there are four basic requirements for an on-line design facility.

a) The provision of a 'suitable' digital computer.

b) The provision of programs to carry out the design calculations.

c) A means whereby the designer can communicate 'efficiently' with the computer.

d) A means whereby the computer can communicate 'effectively' with the designer.

Ad a. What is a 'suitable' computer ? A suitable computer is a machine with sufficient core memory to handle design programs without difficulty. The machine should be able to handle peripheral devices such as line-printers , on-line teletypes and some form of backup storage.

Ad b. The available programs are not a problem except for their compatibility. That is to say that the output from one program must be usable as input for the next program.

Ad. c. Is there an on-line teletype available or a video device ?

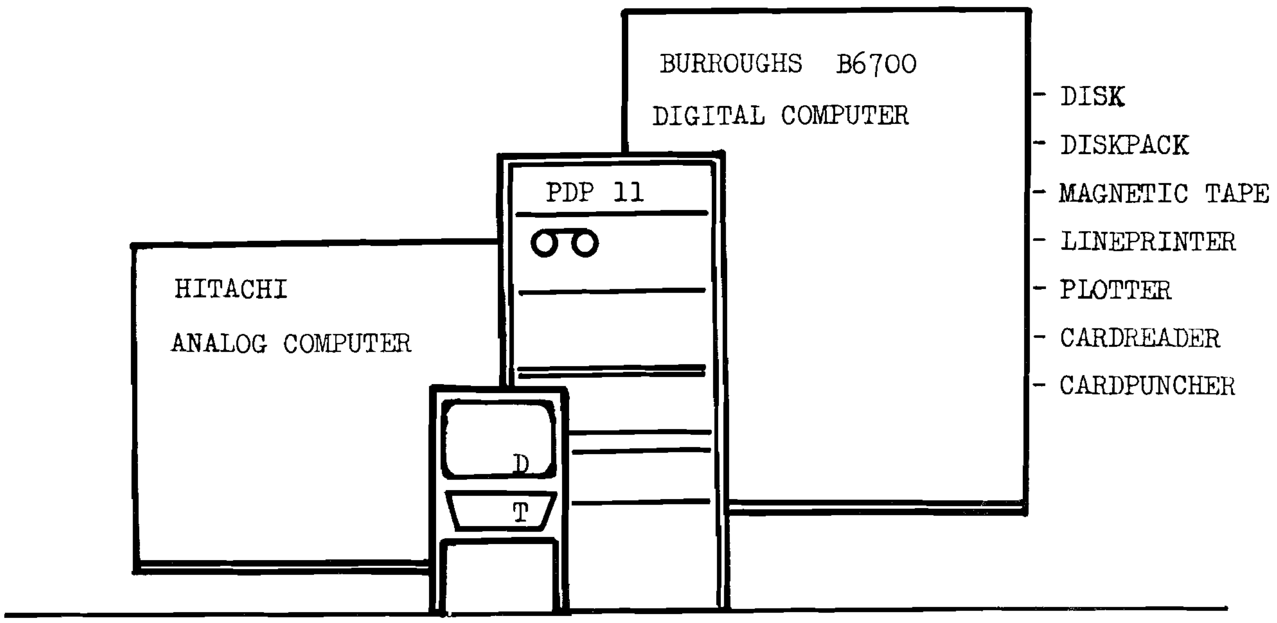
Ad d. How can the computer communicate with the designer ? Via a teletype Video Display? Is a hardcopy possible?

The quality of the man/machine interface will be largely dependent on the hardware devices forming interface and the supporting software. The next section will handle the computer configuration and the software requirements.

1.03 The Proposed Hardware Configuration

The installation of the B6700 Computer at the Mathematical Computing Center of the Technological University of Eindhoven has created a number of new possibilties that previously were not possible but very much desired. Many Departments possess one or more smaller digital or analog computing devices. The digital machines are extremely fast but suffer from the lack of backup storage facilities for the output from these machines. There is also insufficient core memory available for large programs without some form of overlay. The analog machines produce large amounts of numeric output for which no convenient and efficient storage medium is available.

The present hardware configuration permits a smaller computer such as the PDP 11 to communicate with the large B6700 computer. In this way, the smaller computer can use the resources of the larger machine. These resources are two processors of 5 MHz. , central core memory of 164 Kilo byte , fixed disk storage of 100 Mega bytes, exchangable disk storage of 50 Mega bytes, 6 magnetic tape units and such hardcopy devices as lineprinters, plotters cardpunchers.



D= Visual Display

T= Teletype

fig. 1.03 a

1.03

Block Diagram of the Proposed Hardware Configuration

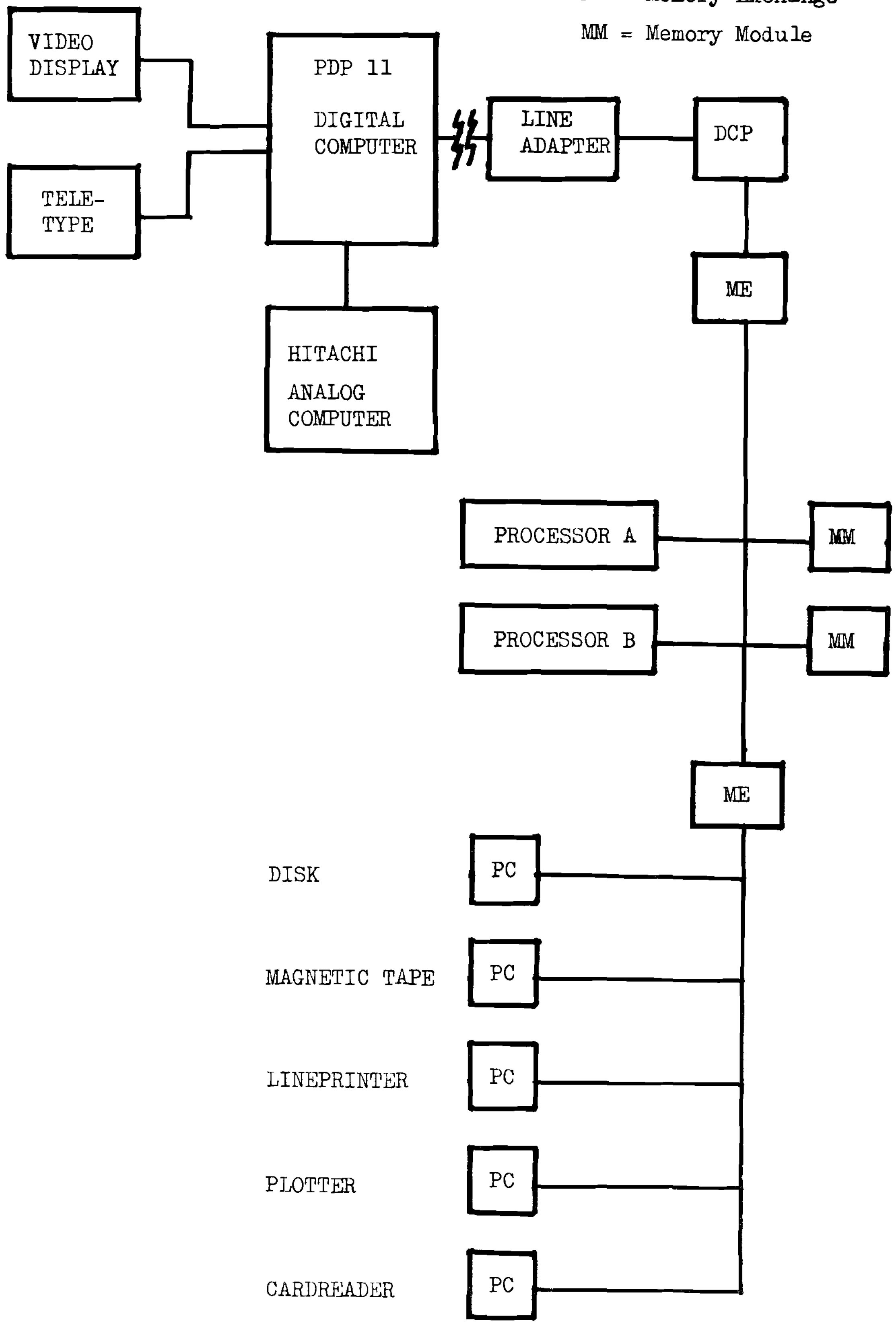
fig. 1.03b

DCP= Datacommunication Processor

PC = Peripheral Controller

ME = Memory Exchange

MM = Memory Module



1.04 The Design Process

In the previous section the design process has been described as an iterative process during which the designer continually makes decisions based on the results of calculations and experience.

This iterative process is represented in the following block diagram.

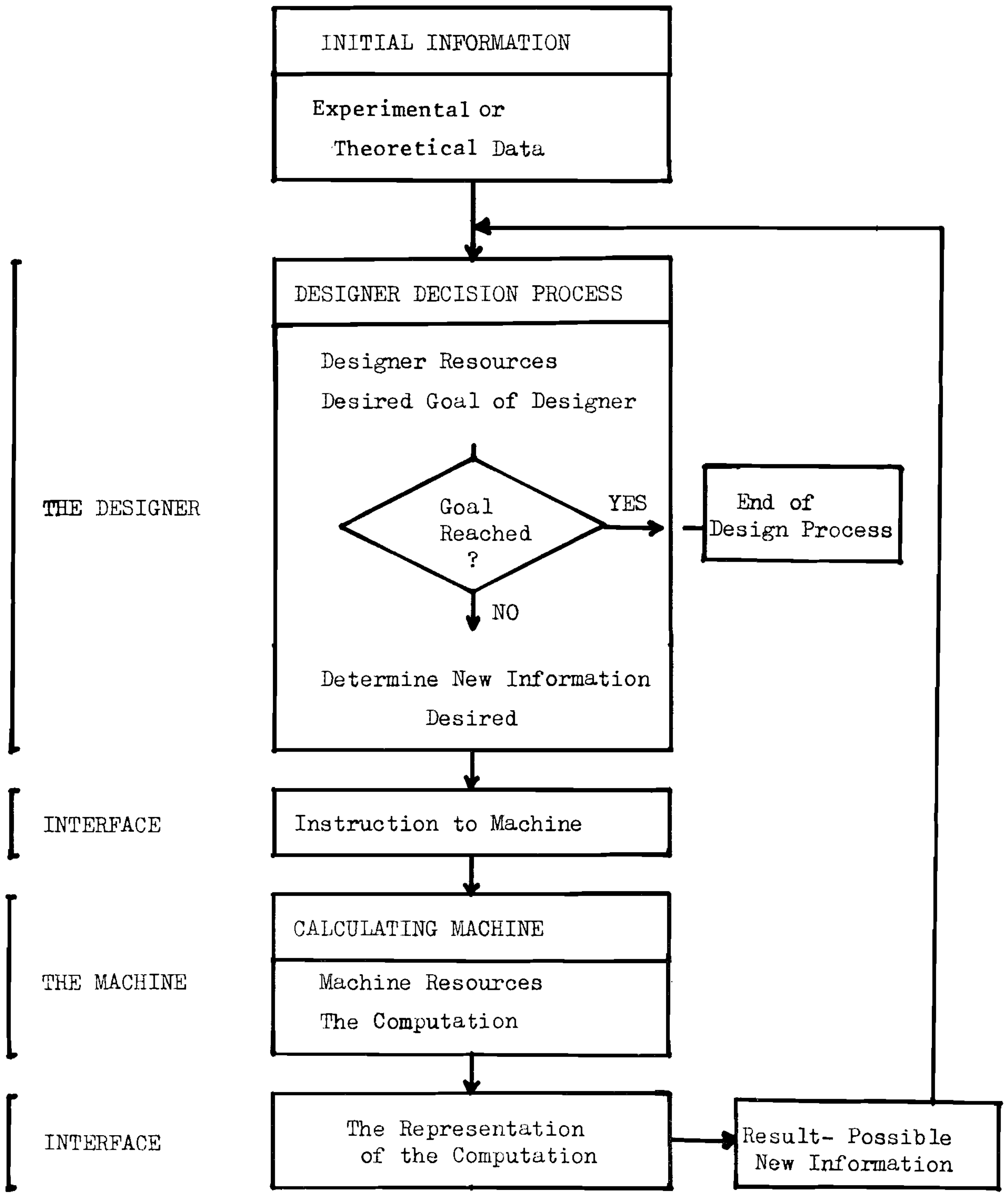


fig. 1.04a

1.04

In the block diagram it is assumed that the numeric calculations are performed by a digital computing machine. The following will be a brief analysis of the design process depicted in fig

The basic components of this process are-

a) <u>The Designer</u> with the resources -Theoretical Knowledge
-Experience
-Insight

b) <u>The Digital Machine</u> with the resources - High Speed Computing Capability
- Large Information Storage Capacity
- Information Handling Capabilties

The two spheres of interaction between the designer and the digital machine are-

a) <u>The communication from the designer to the machine</u>.
How can/does the designer tell the machine what to do i.e. what program is to be executed, what is to be used as input information and how is the output information to be presented.

b) <u>The communication from the machine to the designer</u>.
How can/does the machine tell the designer what it has computed i.e. can the machine represent the computed numeric data in such a manner that the designer can extract the information he desires without undue effort or misinterpretation.

The activities performed by the two components of the design process are-

a) <u>The Designer</u> does - the interpretation of the information produced by the machine.
- the evaluation of the information
- the deciding on possible alternatives
- the issuing of instructions to the machine

b) <u>The Machine</u> does - the numeric computation on the basis of the given instructions
- the conversion of raw numeric output data into the form desired by the designer

1.04

The process goes through a number of cycles until the designer has either found a satisfying solution or he has concluded that-

a) His initial information was invalid or insufficient

b) The design programs are incorrect or insufficient.

It is of course desirable that the number of cycles needed to reach an acceptable solution will be minimal. This will be dependent on the quality of the two main components of the design process i.e. the designer and the programs for the digital machine and the designer/machine interfaces.

The aim of this project is to guide the above described design process in an optimal manner.

By optimal is understood-

a) The efficiency or ease by which the designer can communicate with the machine.

b) The effectiveness of the communication between the machine and the designer. This effectiveness is measured in terms of information representation and the time lag between the issuing of an instruction to the machine and the availability of the desired information to the designer.

1.05 The Dynamic Behaviour of the Design Process

In the previous section the design process has been described and a few desiderata for this process have been formulated. The design process generates a sequence of activities - designer activity- machine activity- designer activity - etc. until an acceptable solution has been found. This sequence of activities is the manifestation of the design process. The activities are detectable by -

a) The instructions given to the machine by the designer
b) The computation performed by the machine
c) The appearance of new information for the designer.

The following paragraphs will attempt to describe this sequence of activities , starting from an initial set of information through to an acceptable solution. In order to facilitate the description of the design process it will be desirable to formulate the general properties of the relationship (s) between the input information to a computer program, the action performed by the computer program and the output from the computer program.

The execution of a computer program can be defined as the mapping or transformation of an initial set of data into a resultant set of data by an algorithm defined by the program text. On the basis of the above concept of mapping, the iterative process by which the designer arrives at the desired results (Section 1.01 or Ref. 14) can be conceived as a sequence of mappings or transformations.

This sequence can be visualized as follows-

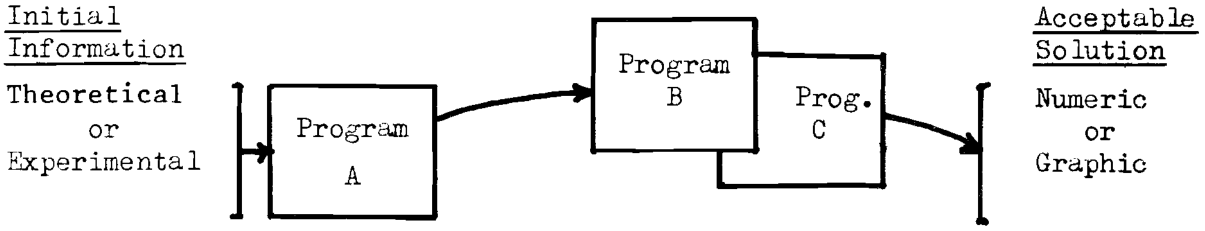


fig. 1.05a

The intitial set of information , either theoretical or experimentally obtained, will undergo a series of transformations until the designer has

1.05

reached an acceptable solution. The sequence can be divided into a number of transitions.
Each transition has the following elements-

a) A transformation algorithm
b) A source i.e. the data which is acted upon by the transformation algorithm
c) A result i.e. the data produced by the transformation algorithm.

A program can now be defined as the implementation of an algorithm which performs a given transformation.
A transition can be described by the following metalinguistic expression.

$\langle$ result $\rangle$:= $\langle$ transformation algorithm $\rangle$ ($\langle$source$\rangle$)

The transformation algorithm will act on the item in parentheses. If the following definition also holds

$\langle$ source $\rangle$ ::= $\langle$ result $\rangle$ or $\langle$ initial data $\rangle$

then a chain of transformations leading to the acceptable solution can be described by

$\langle$ acceptable solution $\rangle$:= $T_n\ T_{n-1}\ T_{n-2}\ \ldots . T_1$ ($\langle$ initial data $\rangle$)

where T_n is the n-th and last transformation and T_1 the first .
In the above , it has been assumed that each transformation will accept as input the output from any previous program. This is in general not the case.

Up to now the influence of the designer on the sequence of transformations and the specific choice of data to be used as source , has not been taken into account. The designer can be expected to follow through a logical sequence of transformations and to use only those source items for a particular transformation which are relevant. However the possibility of an erroneous or delibrate attempt to let a transformation act on data that will produce nonsense will always remain. Whether the transformation on faulty source data is performed or not is a question of implementation. In the following it is assumed that each transformation will accept an arbitary set of data as source.

1.05

In order to analyse the influence of the designer on the sequence of transformations let

a) A,B,C be three programs performing three transformations

b) X_0 be the initial data set information

c) X_{0A} be the result of program A operating on data set X_0

X_{0AB} be the result of program B operating on data set X_{0A} etc.

Then the following graph can be drawn

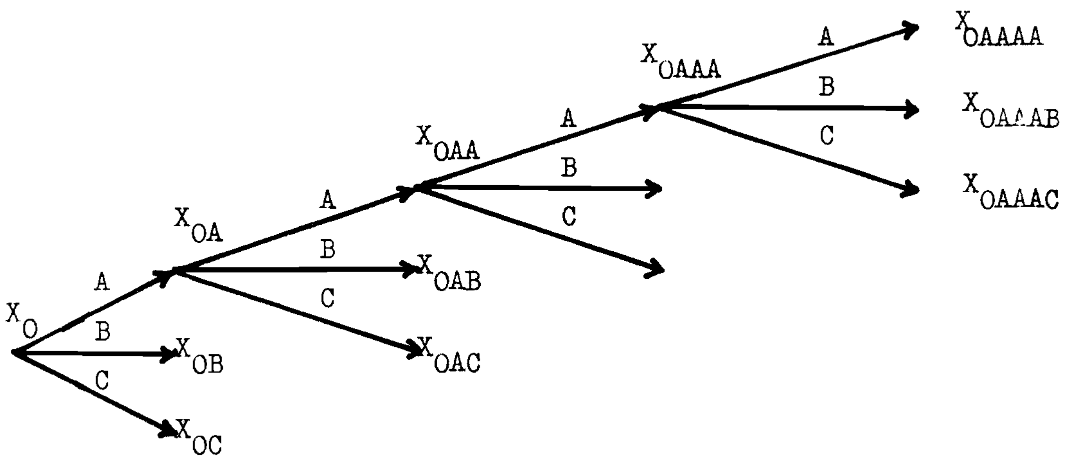


fig. 1.05b

An arrow represents an activity of the machine in performing the algorithm of the transformation. At each node, the designer has the choice of three programs and a number of data sets. The initial data set X_0 can go through a large number of transformations before the designer reaches an acceptable solution. The important thing to note is that the path taken through the graph is not a priori known and therefore the number and manner of creation of the data sets is also unknown. For the time being it has been assumed that each program will accept an arbitrary data set as source. By arbitrary is meant that a program is indifferent to the history of creation of the data set specified as source by the designer. This is not a realistic assumption but will be amended later on.

In general it can be said that at each node the designer determines

a) Which transformation (program) will be next

b) Which data set will be used as source for this transformation

1.05

In fig. b the collection of datasets from which the designer will choose a new source has not been indicated. If for example program A had been activated followed by program B and assuming an initial data set X_0 then on the termination of program B the possible data sets will be-

X_0 , X_{OA} , X_{OAB} and X_{OB} . Therefore the record of activation of the desig process can be said to be the collection of data sets created by the seque of transformation algorithms chosen by the designer.

The situation at each node of the graph can be sketched as follows-

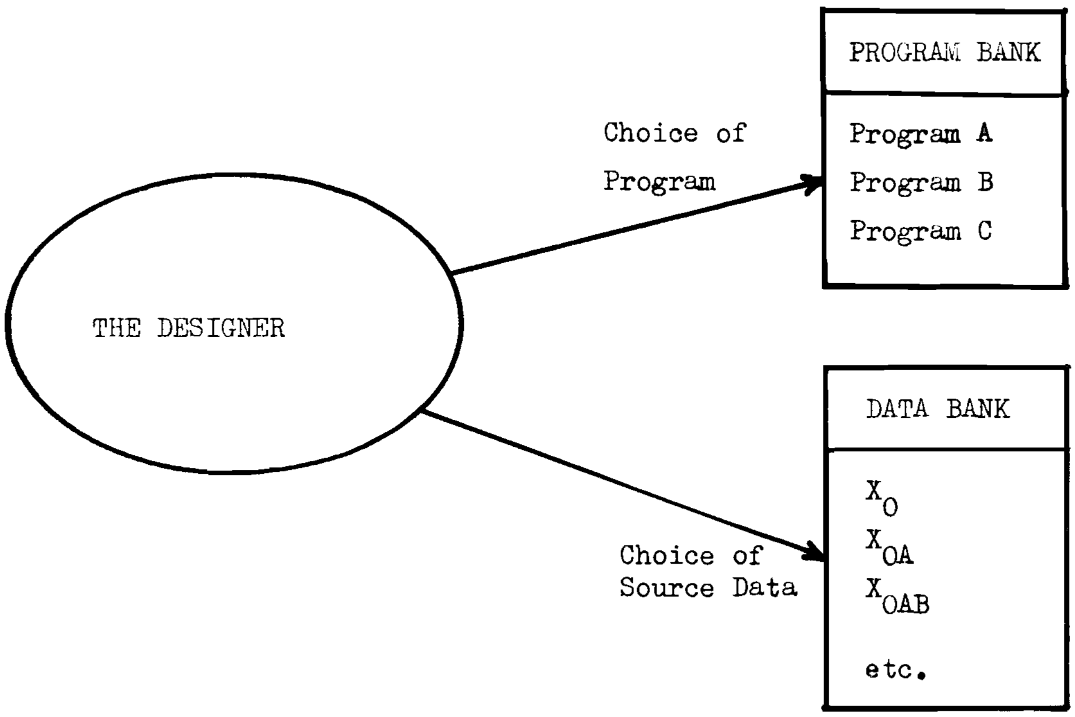


fig. 1.05c

The conclusions of the analysis can be summarized as follows-

a) The designer must be able to indicate to the machine which transformatio algorithm he desires.

b) He must also be able to indicate which data set is to be used as source.

c) The designer would logically choose a particular sequence of programs to act on a meaningful series of data sets but it will be necessary to safeguard the design process from the creation of data sets with no inherent meaning i.e. data sets containing nonsense .

1.06 The Proposed Software Support

From the preceeding sections , the areas in which software support will be desirable for the optimal performance of the design process have become evident.

These areas are-

a) The necessity of an efficient and effective man/machine interface.

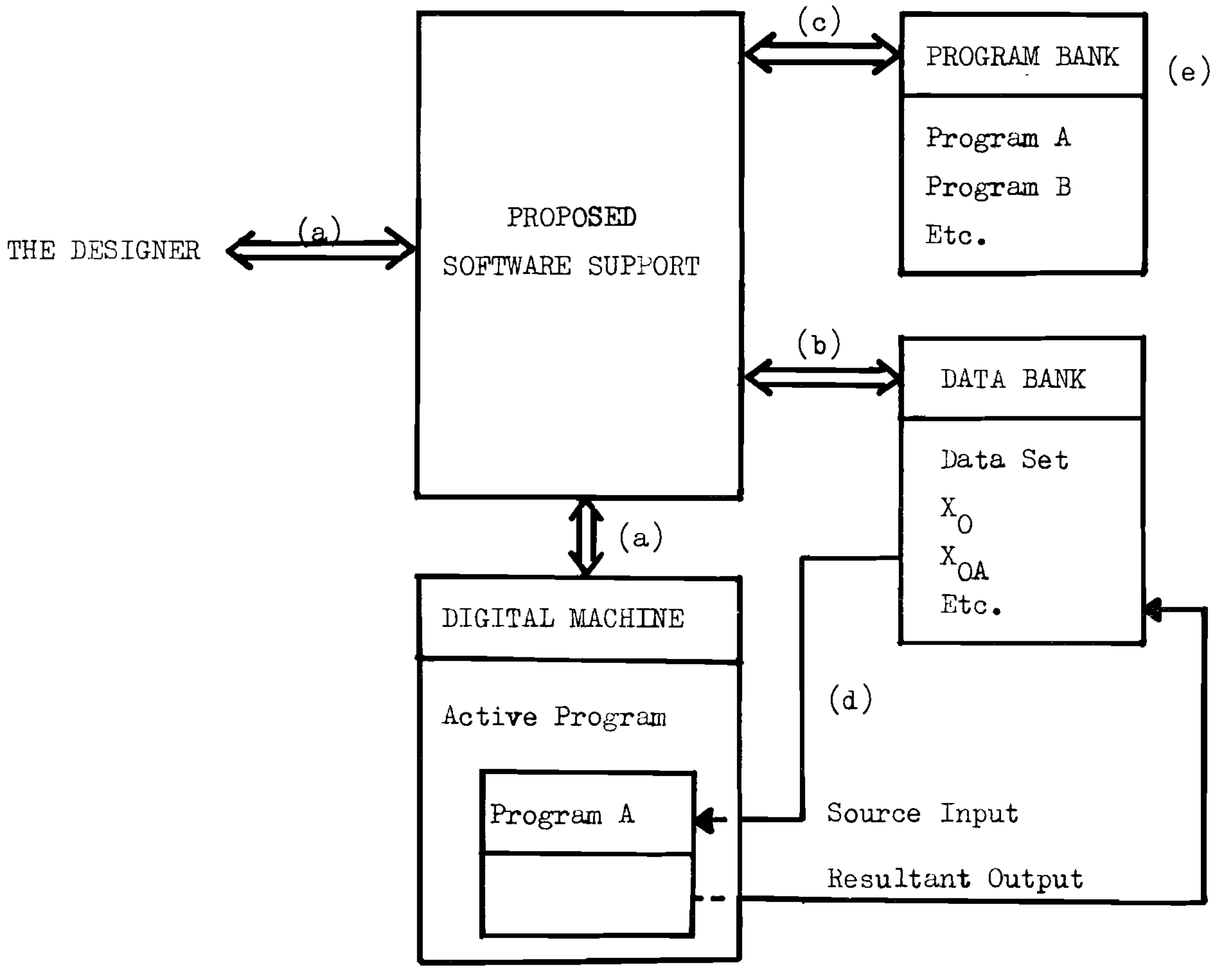
b) The maintenance and logging of data sets created by the designer during the design process.

c) The maintenance of a library containing the design programs.

d) The coupling of a particular data set with the design program desired by the designer.

e) The availability of suitable design programs.

The following diagram will illustrate these issues.



1.06

Each of the above mentioned areas will be handled in more detail in the following paragraphs.

a) <u>The Man/Machine Interface</u>

The first prerequisite for an efficient and effective man/machine interface is an on-line information channel between the designer and the machine. By an on-line information channel is meant that there is no intervening information medium between the designer and the machine. A set of punched cards containing instructions and numeric input data is an example of an intervening information medium . The cards must first be punched and then be read into the cardreader device. There can also be a considerable time lag between the input and the final output. In general an on-line information channel will consist of a teletype or visual display at the designer side of the channel and the necessary input-output supporting software and hardware at the machine side. The designer will then be able to type in his instructions and data directly on the teletype keyboard and recieve the return information from the design program via the teletype line printer. An on-line information channel can therefore be very efficient and effective if the time between an instruction from the designer and the response from the machine is a matter of seconds.

The man/machine communication can be divided into two main categories- from the designer to the machine and from the machine to the designer. The designer/machine communication will consist mainly of instructions to activate design programs . Supplementary instructions demanding information on the presence of design programs and data sets would also be useful to the designer. Designer instructions will require the implementation of an interpreter . This interpreter will convert the instructions into meaningful software entities. The machine/designer communication will consist largely of the numeric or graphic information demanded by the designer. This information is produced by the design programs. Other types of desirable communications to the designer will be error messages on faulty designer instructions and error messages on possible run-time errors in the design programs.

1.06

It is obvious that all designer/machine communication should be straightforward and to a large extent self-evident. This will promote the ease in use and therefore enhance the accessibilty of the design programs.

b) The Maintenance and Logging of Data Sets

Whenever the designer gives an instruction for the execution of a transformation (design program) , a demand for a particular data set to be used as source is created. During the design process a large number of these data sets may be created. In order that the designer can specify and the machine can identify a given data set , each data set must have a unique name. The existence of a particular data set to be used as source must be verified before the desired design program is activated. A list of data sets must be kept and updated. Complications may arise if the resultant data set of a design program is to have the same name as an already existing data set.

The supporting software in this area should consist of-

a) The upkeeping of a directory containing the names of data sets and if necessary on which device they are stored.

b) The verification of the existence of the data set specified by the designer as the source data to the desired design program.

c) A check for ambiguity if the designer is about to create a data set with the same name as an already existing data set.

d) Any other possible operations necessary so that the design program can be activated without error conditions arising.

c) The Maintenance of a library.

The demand for a design program will entail the verification of the existence of the desired program and the preparation necessary for the execution. The designer may also desire information on the availibilty of design programs.

The software support can be expected to perform the following-

a) The maintenance of a directory containing the names of available design programs.

b) The preparation for the execution of the design program.

1.06

d) The Coupling Data Set/ Design Program

Once the existence of a desired design program and indicated data set has been verified , then the design program must have a means of knowing which data set it must use as source. Therefore some provision must be made to couple the fixed data input base to data set indicated by the designer.

e) The Available Design Programs

The primary requirement of each design program will be that it is able to perform the transformation it is intended to perform in the most efficient manner. By efficiency is meant the amount of processor time and core memory it will require. This will depend on the particular algorithm employed and the manner of its implementation.

Each design program will in general demand an input source data set and produce new output data. This output data is destined for the designer. A copy of this data set must be made so that the designer can use it as a source input data to the next transformation . This next transformation could be a transformation creating a new data set or a transformation creating a hard-copy output on the line printer.

It will be desirable to protect each design program from run-time errors such as divide-by-zero etc. which could have a negative effect on the designer.

Once the indicated source data set has been coupled to the design program , there will be the problem of extracting the correct information from the data set . This problem is closely associated with the problem of the structuring of the output data set. This because it has been assumed that a design program will accept an arbitary data set as source . But this source data set could have been created by a previous design program.

The software support in each design program will be

a) Means of extracting the correct information for the computation.

b) The actual computation

c) The handling of run-time errors

d) The creation of the output data set in such a manner that the next design program can extract the information it will need.

1.07 The Proposed Data Structure

In considering the proposed data structure it will be advantageous to make use of those data handling facilities which have been implemented on the B6700. The B6700 is file-oriented in the sense that the user does not have direct access to the actual peripheral device. In general the access to data on physical devices such as cardreaders, disk units and the transference of the data is handled by the Input-Output Subsystem . The I/O Subsystem acts as an interface between the program reading or writing the data and the device containing the actual data.

A file is considered to be a group of related records. Each file has a number of properties called "attributes" . These file attributes are used for

a)	Identification	- name of the file and on which device
b)	Structuring	-maximum record size and units used(words or char)
c)	Status	- to determine if a file exists and is available
d)	Diagnostics	- to determine which errors have occurred during an I/O operation
e)	Security	- to specify how certain files may be used and by whom.

The I/O Subsystem does the searching for a file and manages the transference of data from the program and the physical device. It also maintains a directory for the fixed head-per-track disk and directories for the removable disks. These directories are also accessible to the user.

The following is a functional division of the kinds of files per device type as can be considered relevant to the proposed CEPP configuration.

a) For the On-line Communication

To ensure a reasonable response time in the communication between the designer and the design programs a DATACOM file can be considered essential. DATACOM files are handled by the DATACOMMUNICATIONS PROCESSOR (see Section 1.04 fig 1.04b) and are associated with remote devices such as teletypes and visual displays.

b) For Long Term Storage

There are two possibilities for the long term storage of user data and design programs. These are the removable disk and the magnetic tape . Storage on magnetic tape has the disadvantage that operator intervention is needed to mount the desired tape.

1.07

c) <u>For Temporary Storage</u>

All data files created during a design session can be stored on the fixed head-per-track disk. Files destined for long term storage can be copied to removable disk or magnetic tape in a more efficient format at the end of the session. By a more efficient format is meant that the physical amount of storage space will be minimal.

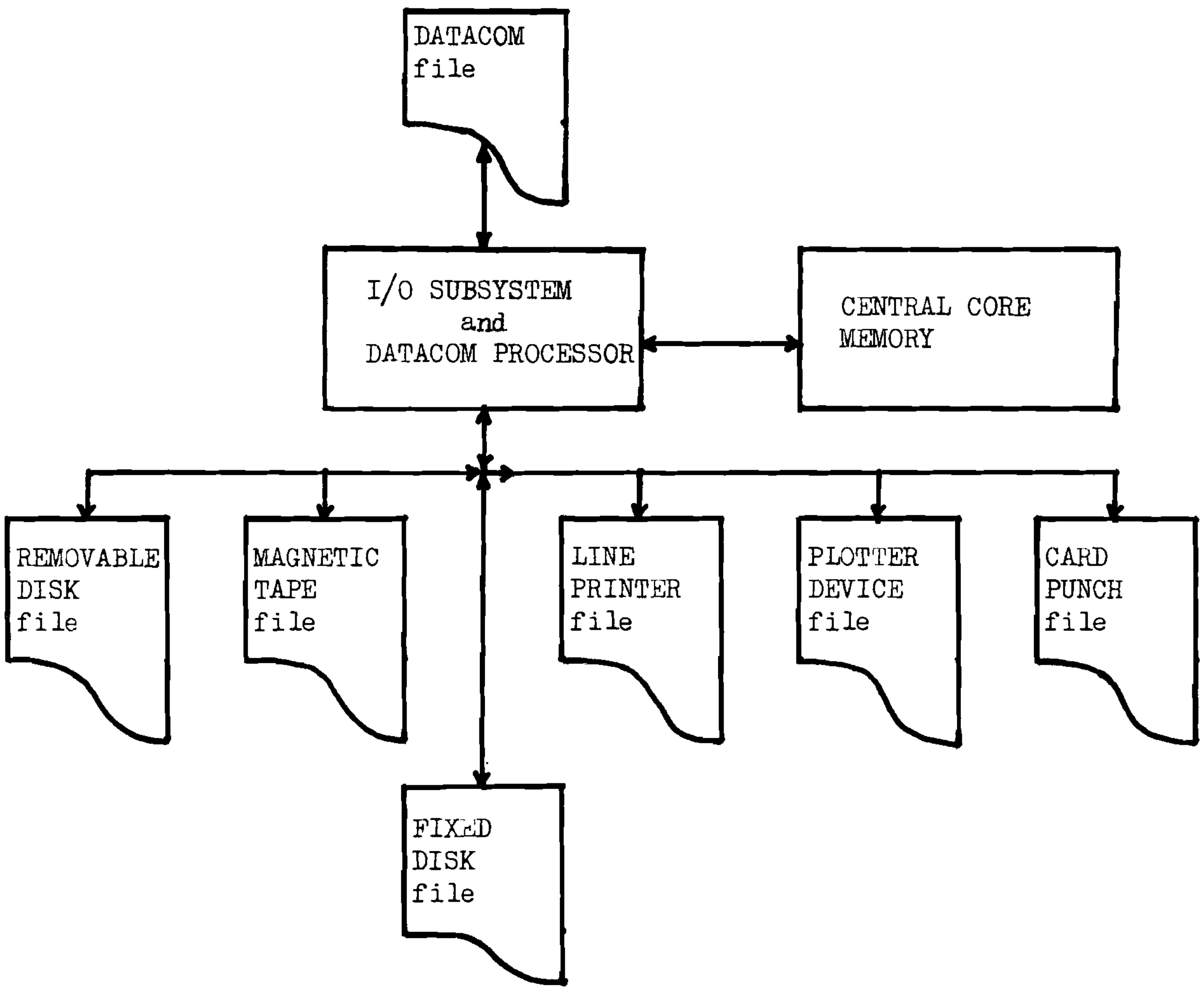
d) <u>For Hardcopy Output</u>

The following forms of hardcopy output can be considered desirable for an effective design process.

1) Numeric and graphic output from a line printer.
2) Graphic information from a plotter device.
3) Hardcopy storage of numeric data from the card puncher or paper tape puncher.

The above mentioned files have been incorporated in the following diagram representing the proposed data structure.

On-line Communication



2.00 Special Topics Concerning the Burroughs B6700

2.01 The B6700 System Organization

The B6700 is a highly structured type of computing machine and is designed with the specific intention of facilitating the execution of ALGOL- like programs. The structure of the B6700 Operating System is based on the ALGOL 60 premise that (static) block structuring is the natural, if not essential, form for the expression of complex algorithms. All B6700 operating system algorithms are written in ALGOL-like languages and are block structured.

The following source card deck for the B6700 will demonstrate this structuring.

```
? JOB OOSTERBAAN ; QUEUE=2  ; USER=U411S417 ;                            ┐
        BEGIN                                                            │
        ? COMPILE E/IL/OOSTERBAAN/TEST WITH ALGOL FOR LIBRARY ┐          │
        ? DATA                                                │          │
             begin                              ┐             │          │
             file  IN(KIND=READER);             │             │          │
             procedure P( A ) ;                 │             │          │
             array  A [*];                      │             │          │
                 begin                          │ ALGOL       │          │ JOB
                 ......                         │ SOURCE      │          │ TASK
                 .....                          │             │ COMPILE  │
                 end;                           │             │ TASK     │
             .. MAIN PROGRAM                    │             │          │
                                                │             │          │
             ..                                 │             │          │
             end.                               ┘             ┘          │
        ? RUN E/IR/OOSTERBAAN/TEST                            ┐          │
        ? DATA IN                                             │          │
          ...                                                 │ RUN      │
              NUMERIC INPUT DATA                              │ TASK     │
          ...                                                 ┘          │
                                                                         │
?END JOB                                                                 ┘
```

2.01

The source deck given on the previous page is an example of a JOB as implemented on the Burroughs B6700. It is writen in an ALGOL-like language called Work Flow Language or in short WFL. The ALGOL-like program structure is clearly visible. The following description of the JOB will convince the reader that WFL language has the same characteristics as normal ALGOL. The WFL compiler i.e. the code program that processes the JOB input deck is true compiler and produces output in JOBCODE in the same way as the ALGOL compiler produces ALGOLCODE .

A JOB is the principal unit of work containing one or more units of work called tasks between the delimiters BEGIN (first) and ?END JOB . The individual tasks are delimited by invocation statements such as ?COMPILE, ?PROCESS and ?RUN . Anything appearing between two tasks invocations or a task invocation and ?END JOB is considered to be declared (implicit) in the former task . The numeric input data set between the RUN and ?END JOB is declared to be of the type DATA and named IN. This data set is therefore local to the task RUN. The source deck contains two tasks each with its local data set. First the ?COMPILE with its local data set containing the the ALGOL source deck and then the ?RUN containing its local data deck. N.B. The complete JOB is also a task.

Not only is the static structure of the WFL JOB ALGOL-like but also the excecution of the JOB has the features of an ALGOL program. There exists a primary system (intrinsic) procedure called RUN (see ref 12 page 116) which initiates and terminates tasks. First the reader should be aware of the fact that there is no basic difference between a compiler that produces executable machine code and a normal user program that produces some form of visible information. In fact the B6700 FORTRAN and ALGOL compilers are written in ALGOL. The machine-encode version of the WFL compiler is called SYSTEM/WFL and the ALGOL compiler SYSTEM/ALGOL. All executable code files process a number of input files(could be none) and produce a number of output files. The B6700 SYSTEM handles all code files in a similar manner.

2.01

The previously illustrated WFL source deck can be split up into the following tasks.

1) Task 1
   Compile all WFL source cards with WFL and call the resultant JOBCODE OOSTERBAAN .

2) Task 2
   RUN the JOBCODE file with the name OOSTERBAAN.

3) Task 3
   Compile the Algol source deck with Algol and call the resultant ALGOLCODE E/ER/OOSTERBAAN .

4) Task 4
   RUN the ALGOLCODE file named E/ER/OOSTERBAAN and use the card images after the control card ?DATA IN as a cardreader file of the name IN.

It can be noted that a Compile task is an implied RUN i.e. COMPILE WITH ALGOL translates into RUN SYSTEM/ALGOL and the COMPILE WITH WFL becomes RUN WFLCOMPILER .

The sequence of the tasks in the execution of the JOB is illustrated in the following diagram.

$$\begin{tikzcd}(1)\ \text{Task 1} \arrow[r] & (2)\ \text{Task 2} \arrow[r] & \Big[\ (3)\ \text{Task 3} \arrow[r] & (4)\ \text{Task 4}\ \Big]\end{tikzcd}$$

Task 2 is initiated by the WFLCOMPILER.
Task 3 and Task 4 together form Task 2.

The WFL JOB text could be replaced by the following BEA source text.

```
procedure RUN( ACODEFILE or APROCEDURE );  specification part + body ;
procedure WFLCOMPILER(WFLMESSAGE);
array  WFLMESSAGE[ * ] ;
begin
translate the contents of the array into JOBCODE.
if OK then RUN( the JOBCODE) ;
end;
RUN (WFLCOMPILER);
```

The above is a simplified illustration of the correspondence between the Algol 60 premise and the Burroughs System Organization. The JOB is converted to a sequence of procedural steps and is structured as a set of nested blocks.

2.01

Assuming that the previously mentioned SYSTEM intrinsic RUN has the following simplified form-

```
procedure RUN ( <executable  code file  name > ,
                <source  input > ,
                <destination output > ) ;


where  <  source input >  ::=  <card images in an array >
                               <data files >
                               < none >

       <destination output > :: =  <executable code file >
                                   <data  file >
                                   < none >
```

then execution of the JOB will have the following operations.

Task 1
RUN (SYSTEM/WFL, card images of JOB deck , CODE of JOB deck);

Task 2
RUN (CODE of JOB deck) ;

The CODE of the JOB deck as produced by the WFL compiler is in JOBCODE and will have the following content but in machine code.

Task 3
RUN (SYSTEM/ALGOL, ALGOL source deck , CODE version of source deck);
Task 4
RUN(CODE of ALGOL source, DATA IN , output files if any);

The above is a simplified illustration of the correspondence between the ALGOL 60 premise and the Burroughs System Organization. The JOB is converted to a sequence of procedural steps and is structured as a set of nested blocks.

2.01

The outer most block is the WFL task which produces two blocks

a) The compile with ALGOL block

b) The execute the ALGOL code block

Since a block defines the scope of the algorithms identifiers and the dynamic resource requirements, each block can be executed without reference to any other block except for those blocks it contains itself. In other words programs like ALGOL procedures can be produced which exhibit strong locality i.e. self supporting.

The result is that also the manifestation of the computing process of each block, that is to say a) The Core Memory requirements
b) The Processor Time requirements
c) The I/O requirements

will also exhibit strong locality.Only the block that defines the scope of the identifiers used in the execution of an instruction in that block needs to be in core memory at the moment of execution.

The realization of the ALGOL 60 premise in the Burroughs System Organization is demonstrated by-

a) All JOBs are block-structured into tasks. Each task contains local entities.

b) The presence of the system intrinsic procedure RUN. This procedure can be understood to be declared beyond the JOB block. The actual parameters of this procedure are the names of executable code files, input information and output information. This type of algorithm is representative for for algorithms generally found in ALGOL.

The bonus derived from this highly structured organization is the possible implementation of "virtual" memory and the possible assessment of the dynamic resource requirements at any particular phase in the execution of a program. This will be handled in the next section.

2.02

The Working Set and Virtual Memory

The B6700 definition of a "JOB" has two components :

a) The time-invariant algorithm.

b) The time-varying data structure which is called the"record of execution " of that algorithm .

The"record of execution"defines at any time

a) the execution state of the job, including the values of all variables;

b) the addressing environment that the processor serving this job may access;

c) the interblock/interprocedure/intertask flow of control history .

What does the "record of execution" need to contain at any given instant of time in order to process a "JOB" effectively ? The answer to this question can be given by considering that all B6700 compilers produce segments of machine code . Each segment is the coded version of a block as described by the syntax of such a block-structured language.
For instance, the ALGOL 60 block delimiters are the pair begin and end and the pair procedure and ";" . For the WORK FLOW LANGUAGE compiler these are JOB,BEGIN, END OF JOB, and any statement containing COMPILE, RUN,PROCESS,CALL , COPY, REMOVE and BIND (there are others) .

The coded blocks are stored as physically separate (not necessarily contiguous) segments. Each segment is pointed to by a descriptor in the so-called segment dictionary. Since the processor can only be active in one particular segment at the time, only those segments that define the contour of the variables used in that particular segment should be present in addressable core. Actually, one should speak of the records of activation of those segments that define the contour(the scope of validity of the variables)of the variables of that segment in which the processor is now active.

2.02

All other segments are present on auxiliary storage such as a DISK backup device. The segment dictionary of a program contains segment descriptors containing-

a) A "presence" bit which is sensed by the hardware address-formation mechanism. If the bit is "on" then the segment referenced is in core memory.

b) An address field.

c) A segment length field.

Whenever a reference is made to a segment in the "record of activation" of the program and if the "presence" bit is "off" i.e. not in core then a hardware interrupt occurs which will delay further execution of the STACK containing the "record of execution". The System intrinsic procedure GETSPACE will then locate the required segment on the auxiliary storage and transfer it to core.
This is the principle of "virtual memory". All core address space is implied and dynamically allocated.

The B6700 implementation of "virtual memory" does not use paging mechanisms to transfer segments to/from core because this does not reflect the information structure of the programs. The allocation of variable-size segments does cost extra processor time but minimizes wasted storage space due to internal fragmentation. However, under adverse conditions it can lead to the phenomenon called "thrashing". This phenomenon occurs when the System is so busy collecting garbage(segments released by terminated programs and segments deallocated by the overlay mechanism) and reallocating segments, that the System cannot accomplish any useful work i.e. process user programs.

The two key System parameters OLAYGOAL and [illegible] and the system demand generated by the user programs will largely influence the amount of "thrashing" that will occur. The parameter OLAYGOAL is used to set the overlay rate in advance and is defined as the percentage of the programs overlayable space that will be removed within a certain time interval. A program's overlayable space is that space that has not been referenced within the last time interval. The parameter AVAILMIN is the percentage of total core storage that must remain available. If the percentage of available storage is less than AVAILMIN then the JOB with the lowest priority is suspended. The system demand created by the user programs can be considered a non-stationary stochastic process so that the set values for CLAYGOAL and AVAILMIN can only roughly reflect the actual characteristics of the user programs. Any serious mismatch will result in the phenomenon called "thrashing".

<u>The Working Set</u>

The implementation of "virtual memory" can lead to very effective core utilization under suitable conditions as mentioned in the previous section. Only those items either code segments or arrays that are actually needed at a certain phase of the computation will be present in core. This brings up the notion of the "Working Set". Mr. Denning in ref. 12 defines a "working set" as "that minimum collection of program segments which must reside in main memory for the process to run efficiently". A programmer can visualize beforehand which items will be in core at certain critical phases of the execution of his algorithm. A programmer can influence the demand on system resources i.e. processor time and core memory requirements by the structureof his algorithm and his programming style.
The key point in Mr. Denning's definition is of course "efficiently" . The working set can be made very small but if certain items are used very frequently then the process will be continually interrupted in order to transfer this item from disk to core if there is a high overlay rate.

Mr. Organick in ref 11 page 34 gives the following advice as a general rule for the contents of the "small working set" .

1) Those segments that are frequently referenced .
2) Structured variables such as <u>files</u> and <u>events</u>.
3) Primary descriptors of <u>arrays</u>.
4) Descriptors of System intrinsics such as I/O procedures and other intrinsics such as <u>wait</u>, <u>cause</u>, <u>procure</u> and <u>liberate</u> .

For a more detailed account of the "Small Working Set " the reader is advised to consult section 3.7 of Computer System Organization by E. Organick (ref 11).

3.01 Task Invocation

On the Burroughs B6700 , a wide range of task invocation statements are possible in Work Flow Language but only three are possible in Burroughs Extended Algol. This due to the fact that most WFL invocation statements are variants of the three basic Algol types. The correspondence will be handled in the following paragraphs.First the three basic types of task invocation statements: -

a) call	invokes a synchronous dependent task
b) process	invokes an asynchronous dependent task
c) run	invokes an asynchronous independent task

An explanation of the terminology used is as follows.
The process that initiates a task is called the initiator. The process that has been invoked is called either a sibling or a partner. By a synchronous dependent task is meant that the initiator will not continue processing while the sibling is active, i.e. the initiator will be suspended until the sibling has become inactive . The initiator must not terminate before the sibling terminates. The same words apply to an asynchronous dependent task except that the initiator may continue processing. The run invokes the task as a completely independent task. Both the initiator and the sibling will be processed at the same time and terminate without influencing each other .

The correspondence between the WFL task invocations and the BEA task invocations can be listed as follows -

In WFL	In BEA
?RUN ‹ filename ›	call ‹ filename ›
?COMPILE ‹ filename › WITH ALGOL	call SYSTEM/ALGOL use ‹ filename › as input file
?PROCESS ‹ filename ›	process ‹ filename ›
?PROCESS RUN ‹ filename ›	run ‹ filename ›

The WFL compiler translates the WFL task invocations into equivalent BEA invocation statements . In principle there is no difference between a task initiated via a WFL JOB or via BEA invocation statements.

3.02

Variables of the Type Task and Task Attributes

Variables of the type task are structured variables and are used to achieve special types of control and monitoring relationships between tasks. They are similar to variables of the type file in that they also have "attributes" which can be assigned or interrogated.

All task "attributes" are set to default values either by the compiler or by the Burroughs System. Not all task "attributes" are accessible to the user but a large subset is accessible either via control cards in WFL or by suitable assignment statements in BEA.

In general, task variables are used to set, log and interrogate the state of a task i.e. active, scheduld, suspended or terminated or to obtain log operational data such as the elapsed time, processor time and I/O time used by a task. The following will be an example in the use of task variables and the assignment of values to task attributes. The correspondence between a JOB in WFL and effectively the same JOB but in BEA will be shown.

Let the JOB be-

```
?JOB TEST;USER=U411S417/HONEYBEE; QUEUE=2;
BEGIN   T(PRIORITY=99);
?IF FILE MY/PROGRAM IS PRESENT THEN
 RUN  MY/PROGRAM [T] ;
?END JOB
```

N.B. Variables in WFL are not declared but implicitly declared by the first usage.

The net result of the JOB will be-
If the file MY/PROGRAM is present on DISK then execute it as a dependent synchronous task with a priority 99 else do nothing. The JOB is then processed by the WFL compiler which will produce the JOBCODE. The JOBCODE will then be executed either as a dependent or an independent task. The program MY/PROGRAM must (in this case) be executed as a dependent task

3.02

because the JOBCODE task must remain active. Otherwise a critical block exit will occur and program MY/PROGRAM will be disabled . This kind of disabling is aptly called "DEATH IN THE FAMILY" . Therefore the RUN statement in the WFL JOB must be interpreted as a BEA call .

The JOB can now be converted to BEA.

```
begin
task T;
procedure DUMMY; external ;
file TEST(FILETYPE=7,KIND=DISK) ;
replace TEST.TITLE by "MY/PROGRAM." ;
if TEST.RESIDENT then
     begin
     replace T.NAME by "MY/PROGRAM." ;
     T.DECLAREDPRIORITY:= 99 ;
     call DUMMY [T] ;
     end;
end .
```

The result of this BEA program text is identical to the WFL JOB. The variables however must be explicitly declared i.e. procedure DUMMY, task T, file TEST .Moreover it can be seen that the assignment of task attributes is accomplished in the same manner as the assignment of file attributes .

Task attributes may be of the type integer,real ,pointer or Boolean .

The assignment of a pointer task attribute to a task variable has the following construct.

```
replace ‹ task identifier › . ‹ pointer task attribute ›

                              by ‹ simple pointer variable › ;
```

The interrogation of a pointer task attribute has a similar form-

```
replace ‹ simple pointer variable › by ‹ task identifier ›.‹ pointer task
                                                             attribute › ;
```

3.02

The two most important pointer task attributes are-

NAME Generally used to assign the TITLE of an executable code file to a task. Can be writen or read.

FILECARDS Is used to assign file declarations and label equations to a task. Write only.

To illustrate the use of the pointer task attribute NAME let the following BEA text be compiled as MY/PROGRAM i.e. the TITLE of the code file will be MY/PROGRAM .

```
begin
file OUT(KIND=PRINTER);
array HELP[0:11] ;
integer L;
replace pointer(HELP) by MYSELF.NAME  ;        % MYSELF task attribute of the
                                               % type task
scan pointer(HELP) for L:72 until EQL "." ;
WRITE(OUT,<"CODE TITLE IS ", A*>, 73-L, pointer(HELP));
end.
```

The result will be the printout -CODE TITLE IS MY/PROGRAM on the lineprinter.

The use of FILECARDS is similar to the file declarations and label equations used in WFL. In WFL the file declarations and file lable equations which follow a task invocation statement are local to the task invocated.and are passed along to the task at RUN time. The above also applies to the assignment of file declarations and label equations to a task via the attribute FILECARDS. The following will illustrate the equivalence between a WFL task invocation and a BEA task invocation.

```
In Work Flow Language             In BEA

?RUN MY/PROGRAM                   task T;
?FILE OUT(KIND=REMOTE)            replace T.NAME by "MY/PROGRAM.";
                                  replace T.FILECARDS by
                                        "FILE OUT(KIND=REMOTE)"4"00";
                                  call APROGRAM [T];     % APROGRAM is
                                                           external
```

The WFL and BEA programs will produce identical results. Program MY/PROGRAM is started up and text "CODE TITLE IS MY/PROGRAM" is printed on a REMOTE device.

3.02

The reader will note that the terminating character of the attribute NAME is "." whereas the terminating character of the attribute FILECARDS is the Hexadecimal character 4"00" .

The assigment of real,integer or Boolean task attributes has the following construct.

<task identifier> . <task attribute> := variable or value of the same type as the task attribute

The interrogation of a task attribute has the construct-

variable of the same type as the task attribute :=
<task identifier> .<task attribute>

For example if T is the task identifier of a task that has been activated then the statement

```
if T.STATUS GTR 0 then T.STATUS:=-1 ;
```

will result in the termination of task T if it has not already terminated.

The task attribute STATUS used above is one of the most important attributes for Inter-Program Communication. The value of STATUS reflects the state of the task .

The values and meanings are-

STATUS	=0	not active
	=1	scheduld i.e.waiting for a processor
	=2	active i.e. awarded a processor
	=3	suspended i.e. waiting for a processor after being active
	=-1	terminated either normal or abnormal

Once a task is active it can be suspended or terminated by assigning the attribute STATUS to 3 or -1 respectively. A task can also be reactivated after it has been suspended.

3.02

Other useful **real** task attributes are :-

STACKNO	Returns the MIX number of an active task or the negative MIX number if the task has terminated. Each task can therefore be uniquely identified.
STOPPOINT	Returns the segment and relative address at which the last arithmetic fault occurred or at which the task was terminated or suspended.
HISTORY	Returns a real value with a bit pattern encoded to determine how and why a task has terminated.
INITIATOR	Returns the relative station number of the REMOTE device from which the task was initiated. Assigning this attribute has the effect that all files of the KIND=REMOTE in a particular task are associated with that station number. For DATACOM only.
ELAPSEDTIME	Returns the total elapsed time since the actual intitiation of the task in multiples of 2.4 microseconds.
PROCESSTIME	Returns the accumulated processor time in multiples of 2.4 micro seconds.
PROCESSIOTIME	Returns the accumulated I/O time in multiples of 2.4 microseconds.

For further information on task attributes see Ref. 9 the section on Inter-Program Communication , Ref. 7 and Ref. 11 .

The last two important **task** attributes to be handled in this section are
EXCEPTIONTASK of the type TASK
EXCEPTIONTEVENT of the type EVENT .

A brief explanation of their use is as follows:-
If task A intitiates task B then task B will be the EXCEPTIONTASK of task A.
If task B undergoes a change in the value of STATUS then the EXCEPTIONEVENT of task A is caused. Task A can be made aware of changes in STATUS of task B by attaching the EXCEPTIONEVENT to a software interrupt (to be handled in section 3.03) or via the wait/waitandreset intrinsics.

3.03

Variables of the Type Event and Software Interrupts

Variables of the type event are structured variables containing two binary switch fields. The first of theseswitches is the "happened" bit and has a " situation-oriented " function. The second is called the" available " bit and has a " resource-oriented " function.

The variables of the type event are used to signal the "happening " of certain events or the " availability " of certain resources between asynchronous task. If, for instance a certain task has completed an important phase of its computation it can notify the other tasks of this event. When used as a " resource-oriented " function , it will allow a task to enter and exit from what Prof. Dijkstra calls a "critical section ". Such a " critical section " could be that two tasks , task A and task B have access to the same data array. Task B readies the contents of the array for task A. While task A is performing certain computations dependent on the contents of the array, task B could be updating the contents of the array. In order to prevent this from happening , task A must set a flag (the "available" bit) that the array is " not available" . When task A has no further use of the contents of the array in question , the flag can be reset to " available "

The B6700 implementation of the variable of the type event has the following storage structure in the activation record of that block in which the event is declared.

event A

The "happened " bit
The wait head
An event wait queue

The interrupt head
An event interrupt queue

The "resource " bit

3.03

The first part of this structure contains the "happened" bit of the "situation-oriented" event. The "situation-oriented" event is used in combination with-

a) the System intrinsics set,reset,wait and cause

b) the System composite intrinsics waitandreset and causeandreset

c) the software interrupt .

The following is a brief sketch of how the implementation works.

The event wait queue contains the STACK NUMBERS (Task attribute STACKNO) of all stacks waiting for that event to happen. Let ANEVENT be a variable of the type event. Whenever a wait statement is entered by a task (wait (ANEVENT) ;) and if the "happened" bit is "off"(NOT HAPPENED) then the stack will be linked to the event wait queue of that event instead of the READY QUEUE . If a stack is linked to the READY QUEUE it can be awarded a processor and become active again. The stack will be re-attached to the READY QUEUE if the event of the event wait queue is caused i.e. cause (ANEVENT) . It is obvious that some other process must cause the event in question.

The event interrupt queue contains the STACK NUMBERS of those stacks that wish to be interrupted whenever the event is changed from "NOT HAPPENED" to "HAPPENED" . An interrupt can be associated with one event only. Any new association with an event will override the old association. An interrupt must first be declared, then attached to an event and enabled. For example-

```
interrupt HANDLEIT;
          begin
          program text
          end ;
attach ANEVENT to HANDLEIT ;
enable HANDLEIT ;
```

If the event ANEVENT is caused (by the task itself or some other task) then the interrupt will be entered and excecuted. Control will return to the next statement following the statement in which the interrupt occurred unless some go to statement is used.

3.03

The following BEA program text should help to illustrate the workings of events, interrupts and the intrinsics wait and cause.

Let task A be.

```
line nr.    begin

            event OK,HELPB;
            array HELPTEXT [0:29] ;
            procedure B( OK, HELPB, HELPTEXT );
            event OK, HELPB;
            array HELPTEXT[* ] ;
            external ;

400         interrupt HELPTASKB ; begin
                                  determine the reason of the interrupt
                                  find some suitable alternative
                                  enter this information in array HELPTEXT
450                               cause(OK);
                                  end ;
500         attach HELPB to HELPTASKB ;
            enable  HELPTASKB ;
600         process B(OK,HELPB,HELPTEXT) ;

            task A does some useful work in this segment


700         wait(OK) ;          % task A must wait for task B before it
                                  can resume processing.
            ....
            ....
            task A does more useful work here

            .....
            .....

            end.
```

3.03

Let task B be.

```
line nr     procedure B(AOK,HELPME,MYINFO) ;
            event AOK,HELPME ;
            array MYINFO[*] ;
            begin

            start processing
            if an abnormal situation arises then
            begin
100         cause (HELPME) ;
200         waitandreset(AOK) ;
            analyse alternatives given by task A as given in array MYINFO
            end

            end of processing-notify task A
300         cause(AOK) ;
            end .
```

A brief explanation of the above BEA program text is as follows.
Task A is to initiate task B as an asynchronous dependent task. Then it should commence processing until it has reached line 700 containing the wait(OK) statement. While processing task A can be interrupted by task B. In that case the interrupt declared on line 400 will be entered. The reasons for the interrupt will be determined, alternatives will be found if possible and stored in array HELPTEXT. Task B is waiting for the event OK (the actual parameter) to HAPPEN at line 200 of task B. The event OK is caused and reset to NOT HAPPENED because the event OK is also used in line 700 of task A. Task B will come out of the event wait queue of event OK and excecut the next statement after line 200 of task B. Task A will return to the next statement following the statement where it was interrupted and continue processing until it reaches line 700 or it is interrupted again.

3.03

Only one complication can arise. If task A has arrived at line 700 and the event OK is in the "NOT HAPPENED" state then task A will be linked to the event wait queue of event OK. But now an interrupt occurs. How can task A be made active again, so that it can service the interrupt ? This difficulty will be handled in the section " Interrupting a Sleeping Task".

As already mentioned in this section, events can also be "resource-oriented". The system implemented intrinsics procure, liberate and fix are used to coordinate the entries and exits of tasks to/from "critical sections". Let EVT be the event identifier. The procure(EVT) will force any task that is trying to change the state of the "resource" bit of event EVT to "NOT AVAILABLE" , into the event wait queue of event EVT if the bit has already been set to "NOT AVAILABLE" by some other task. If some other task excecutes the liberate(EVT) then the "resource" bit will be set to "AVAILABLE" and all tasks in the event wait queue of event EVT will be linked to the READY QUEUE. If the "resource" bit was "AVAILABLE" then the bit will be set to "NOT AVAILABLE" and the task will resume processing.

The fix(EVT)is a Boolean function intrinsic and can be considered a type of conditional procure . If the event EVT is "AVAILABLE" then fix will return the value:= false and set the "resource" bit to "NOT AVAILABLE". Had the event EVT been in the state of "NOT AVAILABLE" then fix would have returned the value:= true and left the bit unchanged. This can be important because any task trying to enter a "critical section" will be forced into the event wait queue of that event whereas it could be busy processing data which is not dependent on information pertainable in the "critical section". The task could from time to time try to gain access to those resources which have hitherto been " NOT AVAILABLE".
A simple example could be: if fix(EVT) then S_1 else S_2 ;
If EVT is "AVAILABLE" then statement S_2 will be excecuted and statement S_1 if the event EVT is "NOT AVAILABLE".

4.00 Special Software Constructs

4.01 Interrupting a Sleeping Task

This section will be concerned with the problem of waking a sleeping task, so that it can service an interrupt. The subject has been mentioned briefly in the section 3.03 . The solution offered here is essentially the solution offered by Elliot Organick (see Ref.11) .

The problem can be stated as follows-

Let task A be waiting on event X .
Let task A have a software interrupt that is attached to event Y .
Let task B as an asynchronous dependent task of task A cause event Y.

Desired is -
a) That the software interrupt of task A is service i.e. executed .
b) That task A returns to the previous wait condition after the interrupt.

The above outlined problem occurs at line 700 of the program text in section 3.03. Task B has not finished processing therefore task A enters the <u>wait</u> . Task A will remain in this <u>wait</u> until the <u>event</u> OK is <u>caused</u> . In the meantime some abnormal condition may arise in task B and event HEPLB (actual parameter) will be <u>caused</u> . Task B will then wait on event OK . The interrupt of task A however can not be service unless task A is active. Therefore both tasks will wait for event OK to be caused. This undesirable situation can be remedied by the fact that an event can be associated with its wait queue and with its interrupt queue simultaneously .

Therefore if the statement
<u>wait</u> (OK,HELPB) were to replace <u>wait</u>(OK) in line 700
then task A would come out of the wait queue of event HELPB and consequently captured by the interrupt . After the execution of the interrupt , the next statement following the <u>wait</u> will be executed. It is however desired that task A returns to the <u>wait</u> after the execution of the interrupt . In other words a conditional type of <u>wait</u> is needed i.e. if event HELPB is <u>caused</u> then service the interrupt and return to the <u>wait</u> else if event OK is <u>caused</u> then come out of the <u>wait</u> and execute the next statement.

The solution to the problem is based on the implemented complex <u>wait</u> and the use of a dummy statement.

4.01

The following will be an explanation of the complex wait and its use in the solution of the problem described.

The wait intrinsic is implemented as an integer function . The value returned depends on the order of the events given in the event wait list and the particular event caused.

Let EVT1,EVT2,EVT3 be variables of the type event .
Let EVENTNUMBER be a variable of the type integer .
Let the statement
EVENTNUMBER:= wait(EVT1,EVT2,EVT3) ; if EVENTNUMBER EQL 1 then... else..;
appear in the program text.
If event EVT1 is caused then the wait function will return the value = 1 and come out of the wait. Similarly,if event EVT2 had been caused then the value = 2 would have been returned.

The net effect of such a construct is that it is now known which event has caused the wait to be left.
If EVT1 had been attached to an interrupt then EVENTNUMBER would have the value =1 and the interrupt would be executed . However after the interrupt the next statement- the if clause would be executed instead of returning to the wait .

The following construct has been devised.

while wait (EVT1,EVT2,EVT3) EQL 1 do S_0 ; S_1 ;

Here S_0 is a dummy statement and event EVT1 is assumed to be attached to an interrupt.

If event EVT1 is caused then wait intrinsic will return the value = 1 but it will be immediately captured by the interrupt . On return from the interrupt statement S_0 will be executed and the while clause evaluated . The while clause is true therefore the wait will be re-entered. If the while had been false then statement S_1 would be executed . This is the case if either event EVT2 or EVT3 had been caused . The while clause would be evaluated on leaving the wait as being false and therefore statement S_1 would be executed.

The desired construct at line 700 in the previous section will be

while wait (HELPB,OK) EQL 1 do ;

4.02 Separately Compiled Procedures

The Burroughs Algol Compiler will accept any block as suitable for compilation.

A block has two forms.

a) <u>begin</u>
   declarations
   statements
   ...
   <u>end</u>.

b) type <u>procedure</u> (formal parameter list)
   <u>value</u> identifier list
   specification part
   statement ; or .

A block is a statement that groups one or more declarations and statements into a logical entity. A statement however can also be a block.

Normal user programs are blocks of the type (a) with one or more blocks of the type <u>procedure</u> but the statement is terminated by the semi-colon. If a program is compiled as a block of type (a) it will not be possible to pass actual parameters to the program. Since it will be more than necessary to be able to pass parameters from one program to another in the proposed CEPP configuration , this section will be devoted to how and why this can be done.

It must first be shown that it can be done. The Burroughs System Utility SYSTEM/DUMPALL can be considered a prime example.
A user could issue the task invocation in Work Flow Language

?RUN SYSTEM/DUMPALL("TEACH")

The user will then receive instructions via the lineprinter as how to use SYSTEM/DUMPALL . The string "TEACH" is passed to the program SYSTEM/ dumpall as actual parameter. The string will then be analysed and interpreted to mean that the user wants information on how to use SYSTEM/DUMPALL.

4.02

The BEA source deck for SYSTEM/DUMPALL would have the following text-

```
?COMPILE SYSTEM/BURROUGHS WITH ALGOL FOR LIBRARY

?DATA
procedure DUMPALL (A) ;
array A [*] ;
begin
pointer PA;
file OUT(KIND=PRINTER);
PA:=pointer(A);
if PA EQL "TEACH" for 5 then
                        begin
                        WRITE(OUT, The teach information );
                        end
                        else
                        begin
                        scan for other valid commands
                        if found then act accordingly
                        else WRITE(OUT, Error message to user )
                        end;

end. % END OF DUMPALL
```

The last identifier of the program filename (in this case BURROUGHS) is replaced by the procedure identifier by the ALGOL COMPILER . The net result of the above source deck is that the procedure DUMPALL is compiled with the CODE filename SYSTEM/DUMPALL .

The important thing to note is that the lower bound of array A is not specified but given by the asteriks . Also array A is one dimemsional. The activation of the procedure DUMPALL with the literal "TEACH" as actual parameter is only possible via a WFL task invocation statement. In BEA it is not possible to call a procedure with an array as parameter by value because an array has no inherent value.

All parameters of procedures to be activated by WFL task invocation statements must be called by value. Since structured variables such as task, event and file do not have values they can not be passed as parameters.

4.02

The special exception, as already mentioned, was the <u>array</u> with a literal as actual parameter.

Programmatically activated (in BEA) separately compiled procedures may have parameters that are either call by name or call by value depending on the type of parameter and the use intended.

The reasons for desiring a program compiled as a separate procedure can be enumerated as follows-

a) Passing run-time information to programs initiated via WFL task invocation statements. SYSTEM/DUMPALL has been given as an example.

b) Because separately compiled procedures are independent executable program units they can be activated by BEA task invocation statements. A small program segment containing the task invocation statement can activate a very large program and pass along the desired parameters. Moreover, the same program segment can be used to activate different programs in succession.

c) If it is intended to build up a library of programs then each new addition can be debugged before it is added.

d) Each new addition will not nescessitate the re-compilation of the whole. The program doing the activation needs only to be notified of its existence.

e) Only those programs or program segments needed are present in core memory and on the fixed head-per-track disk. Other programs can be stored on a removable disk and called up as needed.

d) A number of users can share the same program segments if they work under the same USERCODE .

Whether or not one is to compile a program or a set of programs as separate procedures will depend on the use intended. Will it be necessary to pass parameters to the program ? If so then a separately compiled procedure will offer a solution.

4.02

The general idea has been given in those passages as to how and why programs could and should be compiled as separate procedures. The following will be a brief outline of the actual implementation .

The first consideration is that only untyped procedure s may be compiled separately.
An untyped procedure has the general form:

```
procedure  procedure identifier  ( formal parameter list ) ;
value  identifier list  ;
  specification part    ;
begin

                the procedure body which is a statement  which may be
                a block.


end.
```

The main program that will initiate the above generalized procedure will then be

```
begin
task T; pointer PA ; array HELP [0:29] ;
  declarations of the variables used in the formal parameter list
    of the procedure ANYPROGRAM
procedure ANYPROGRAM ( formal parameter list ) ;
value  identifier list   ;
 specification part   ;
external ;                      i.e. the procedure body of procedure ANYPROGRAM
                                     will become the procedure body(actual program)
                                     of the separately compiled procedure
....
replace T.NAME by PA;
...
replace T.FILECARDS by PA ;
```

4.02

```
process ANYPROGRAM( actual parameter list ) [T] ;
...
...
end.
```

The task attribute NAME will contain the code filename of the separately compiled procedure .

The task attribute FILECARDS will contain file equations and declarations.

The only part that remains will be the contents of the formal parameter list. What should it contain? . Obviously the contents will depend on the desired relationship between the main program and the separately compiled procedure that is to say the user requested program. Further details are worked out in the section " Implementation of the CEPF/SUPERVISOR" .

5.00 The Realization of the Control Engineering Programming Package

5.01 The Design of the Software Support

In sections 1.04 and 1.05 the design process has been outlined and those areas considered appropriate for software support have been indicated. Some potential software tools have been described in sections
What remains is the actual designing of the software support.

In the first instance , the software support to be designed is directly connected with the overal desired end result. Once the desired end result has been described and analysed into components, then those entities which will contribute to the properties of the end result can be described. The functions performed by the entities and their interrelationships will together define the structure of the software support. The realization will be dependent on the available software and hardware resources.

The desired end result will be to guide the designer to an acceptable solution in an efficient and effective manner. By efficient is meant that the time and energy to be invested by the designer will be minimal and by effective is meant that the design process is simulating and instructive. The efficiency can be advanced by

a) An on-line communication channel between the designer and the digital machine. This will mean that the access to the design programms will be simple and direct. The design information desired will also become immediately available to the designer if a reasonable reponse time is assumed.

b) The availability of suitable design programs and conversion programs. A design program can be said to be suitable if it can accomplish the desired computation with a minimum of processor and I/O time and present the output information to the designer in a concise manner. The availability of conversion programs will permit the designer to enter his input data in different forms. For instance , a transfer function can be defined by the coefficients of polynomials or by the roots of the factored polynomials. Both of these input forms are desirable.

5.01

c) General software support in routine operations and in complex situations. Recurring operations such as the finding data sets and the verification of the existence of a design program must be part of the software support. Complex situations such as the activation of the design program and the coupling of a data set with that program must also be done by the supporting software. Possible error conditions and reasons must also be made know to the designer.

The effectiveness can be promoted by

a) The on-line communication because the concentration of the designer on the problem at hand has not been diminished by the time interval between question and answer.

b) The availability of graphic aids . The most desirable would be in the form of a video display but a hardcopy graphic aid such as a plotter device could suffice.

c) The possibility of obtaining a hardcopy of the results of the computations on a lineprinter. The designer may wish to keep a permanent record of certain results or wish to study the numeric output data at his convenience.

d) The possibility of some form of control over the execution of a design program. The designer might wish to halt a program and desire to know how far the program has progessed and what it has produced. On the basis of this information he could decide whether to continue the program instead of being forced to wait until the final result is produced .

e) The possibility of posing questions on the availabilty of programs or the existence of data sets. Also information on processor time and I/O time could be useful to the designer.

5.01

The above mentioned points can be listed briefly as-

a) Facility
   An on-line communication between the designer and the machine will promote the ease of access and a quick response.

b) The provision of suitable design programs and conversion programs.

c) The surveillance of all operations in case of errors.

d) The flexibility of handling during the execution of a design program.

e) The utility offering the designer useful information.

f) The assistence in routine or difficult operations.

These points roughly define the operations to be performed by the supporting software.The available tools in Burroughs Extended Algol can be enumerated as follows-

a) Structured variables of the type file
b) Variables of the type pointer
c) Structured variables of the type task
d) Task invocation statements
e) Separately compiled procedures
f) Facilities offered by the I/O Subsystem accessible in BEA
g) Other possible facilties such as TIME intrinsics etc.

The basic operation to be performed by the supporting software will be the actualization of the concept of mapping introduced in section 1.05 . This basic operation will provide the framework for all other operations. All other operations,in effect,will support the basic operation of mapping in order to ensure the the overal software support possesses the desired characteristics.

5.02 A General Description of the Implemented Software Support

This section will describe the basic program units of the software support. Each unit will be handled in more detail in section 6.00 .

The CEPP configuration consists of three basic program units and a number of design programs. The program units are-

a)	SYSTEM/CEPP	The preliminary program
b)	SUPERVISOR	The main CEPP program
c)	INPUTSYSTEM	The auxiliary program to the SUPERVISOR.

The design programs can be any desired program such as NYQUIST,ROOTLOCUS as long as these programs have the correct procedure headings so that they can be initiated by the SUPERVISOR.

The overall performance of the CEPP configuration can be split up in the following main phases.

1) The preparation of the CEPP user session.
   The SYSTEM/CEPP program prepares a WFL JOB for the session on the basis of user's USERCODE/PASSWORD. This JOB contains the following items-
   a) COPY from PACK statements for necessary code files.
   b) RUN the SUPERVISOR statement
   c) RUN a diagnostic program called MESSAGE if certain code files are missing.
   d) REMOVE all CEPP code files and user data files at the end of the session .

   This WFL JOB is compiled by the WFLCOMPILER and the resultant JOBCODE is either processed or run by the SYSTEM/CEPP. If the JOBCODE is run then the CEPP user session will remain within the previous JOB otherwise a new JOB will be created. Also if the JOBCODE is processed then the SYSTEM/CEPP will remain active (in a waitandreset) until the user session has ended.The net result of the SYSTEM/CEPP program is that all relevant code files are copied from PACK to DISK and the SUPERVISOR program is initiated within a WFL JOB as a dependent synchronous process. If errors occur then the user will by notified by SYSTEM/CEPP .

5.02

2) <u>The activation of the main program the SUPERVISOR</u>
As already mentioned the SUPERVISOR is initiated by the SYSTEM/-CEPP program as a component of a WFL JOB. The SUPERVISOR is responsible for the proper functioning of the user design programs and the INPUTSYSTEM. It should be aware of abnormal conditions and if possible correct these. The operations to be performed by the SUPERVISOR can be enumerated as follows-

a) The initialisation of the INPUT/OUTPUT files. The INPUT file is either of the KIND READER or the KIND REMOTE whereas the OUTPUT file is of the KIND PRINTER or REMOTE if the program SYSTEM/CEPP is initiated via a CARDREADER or via a REMOTE station respectively.

b) The activation of the INPUTSYSTEM and user design programs.

c) The monitoring of the STATUS of the INPUTSYSTEM and design programs.

d) The co-ordination of the flow of control between design programs and the INPUTSYSTEM in the case of user entered control commands.

e) The provision of suitable diagnostics whenever error conditions occur.

The SUPERVISOR is initiated as a dependent synchronous process within the WFL JOB created by the SYSTEM/CEPP program. The SUPERVISOR in turn starts up the INPUTSYSTEM and design programs as dependent asynchronous processes. This has been done in order to permit a design program and at the same time the INPUTSYSTEM to be active. Also the SUPERVISOR must be able to monitor the STATUS of both the INPUTSYSTEM and the design program. The three processes the SUPERVISOR , the INPUTSYSTEM and the user design program are parallel processes. This solution has been implemented in order to permit instructions from the designer, to be entered via the INPUTSYSTEM or via the design program. This based on the constraint that only one INPUT file of the KIND REMOTE is permitted. Therefore the input must be switched from the INPUT-SYSTEM to the design program and vice versa .

5.02

3) The activation of the INPUTSYSTEM

The INPUTSYSTEM is initiated by the SUPERVISOR as a dependent asynchronous process. Its main function is to serve as an interface between the CEPP user and the SUPERVISOR . As an interface, it will interpret all user instructions into meaningful software statements or constructs. In some cases the INPUTSYSTEM will handle the interpretted user instructions otherwise the SUPERVISOR will perform the desired operation. The INPUTSYSTEM will also handle user requests for information concerning the available CEPP programs, the user data file content and other log operational data such as ELAPSEDTIME,PROCESSTIME and IOTIME . The operations to be performed are as follows-

1) Initiate the procedure DATACOM as a dependent asynchronous process.
2) Handle all user instructions as recieved from process DATACOM.
3) If necessary it will copy code files and user data files from PACK.
4) Supply the SUPERVISOR with meaningful software data for the initiation of user requested design programs.
5) Supply the user with information on the availability of CEPP design programs and log operational data.

The procedure DATACOM is initiated as a dependent process because only this procedure has access to the REMOTE input file. By giving this asynchronous process the task of reading the input file , the INPUTSYSTEM will always be accessible for user instructions. The process DATACOM will determine by the contents of the instuction and the state of the INPUTSYSTEM if the message is to be passed to the INPUTSYSTEM. There are three types of user instructions - the request for a design program, the request for information and the control command. These instructions are handled in the section on the INPUTSYSTEM and in the Appendix A .

5.02

4) The activation of the design program

All design programs are intiated by the SUPERVISOR as dependent asynchronous processes..The design programs must have identical procedure headings if they are to be initiated via the CEPP configuration. Existing design program blocks or procedures can be accomodated by the addition/alteration of the CEPP standard procedure heading. The operations to be performed by the design programs are as follows-

a) Initiate a dependent asynchronous process to serve as an INPUT facility. The design program will then remain accessible at all times.

b) Do the actual design computation.

c) Give the user diagnostic messages in the case of errors or missing data.

The INPUT data to design programs will come from DISK or PACK files. Data,if created during the present session then the data will be present on DISK. If the data is from a previous session then the data will be on PACK. Local INPUT data i.e. data needed for this computation only will be given via the REMOTE input file. Each design program will produce only one output file on DISK. This file will contain sufficient information for subsequent programs to decipher the contents. Let the following serve as example .
Let a design program compute the values of the real and imag. parts of a given transfer function for a given frequency range and frequency increment. The data concerning the numeric values of the coefficients of the transfer function have been created during the present session and are therefore present on DISK. The local information will be the desired frequency range and frequency increment . This local information will be given via the REMOTE input file. The output from the design program will be stored on DISK . The original data set containing the coefficients of the transfer function has been mapped into a new data set containing the values of the real and imaginary parts of the transfer function in the desired frequency range.

5.02

5) The termination of the CEPP user session

The user can end the session whenever desired.If the user wishes to store one or more of the data files created during the session then he can give the appropriate commands to copy the desired data files to PACK. From the point of view of disk hygiene it is desirable that not a trace is left on DISK of the user session. This will entail the removal of all CEPP code files used during the session and the removal of the user data files . The operations to be performed are as follows-

a) Copy those data files which the user desires to keep from DISK to PACK .

b) Remove all CEPP code files from DISK.

c) Remove user data files from DISK.

The reason that new data files are first created on DISK and then copying to PACK instead of directly creating the file on PACK is based on the following considerations-

a) PACK storage economy. The actual size of the user data files are not known before they are closed at the end of the session. The file attributes of DISK files are set be default by the I/O Subsytem . When the DISK file is copied to PACK it can be copied with the most economical (minimum storage space) file attribute combination.

b) Not all user data files on DISK are expected to become permanent files(worthwhile storing) .

c) Disk files can be protected from System failure using the file attribute PROTECTION. Setting the attribute PROTECTION=PROTECTED will make it possible to find the last valid block written in the event of a HALT/LOAD

5.02

The following is a block diagram of the CEPP configuration in the situation where the JOBCODE has been processed and a design program is active.

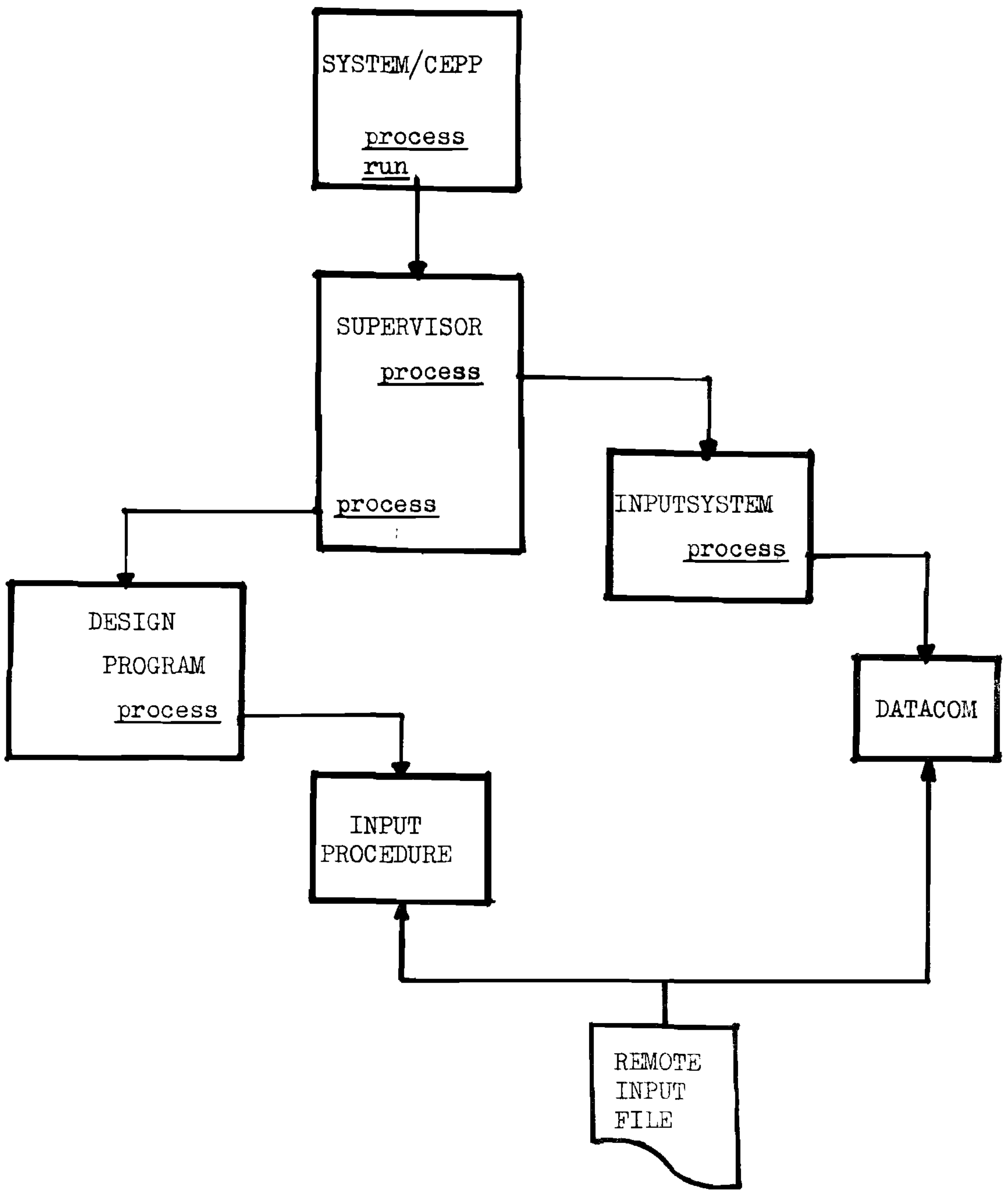


fig. 5.02 a

6.00 <u>The Program Units of the Software Support</u>

6.01 <u>SYSTEM/CEPP</u>

The purpose of SYSTEM/CEPP is to handle preliminaries and to initiate the CONTROL ENGINEERING PROGRAMMING PACKAGE . These preliminaries consist of USERCODE/PASSWORD verification, assignment of options and the preparation and creation of a WFL JOB containing the necessary WFL statements to start up the PACKAGE .

If a potential user has been accepted as being a valid user then SYSTEM/CEPP will create a WFL JOB.
The WFL JOB will be created on the basis of -

a) The USERCODE/PASSWORD combination given by the designer
b) The built-in USERCODE/PASSWORD combination of the PACKAGE
c) The present MIX NUMBER of SYSTEM/CEPP (STACKNO)
d) Whether the JOB is to be activated as an independent or a dependent asynchronous process .

The essential elements of the WFL JOB are-

a) COPY statements to copy the necessary program units from storage
b) A RUN statement to activate the SUPERVISOR
c) A REMOVE statement to REMOVE all CEPP CODE files and user DATA files at the end of the JOB .

The rationale of the two types of JOB tasks i.e. independent versus dependent is as follows.

Whenever a JOBCODE file is initiated as an independent asynchronous process, the process as a whole is monitored by the MCP (Master Control Program) . The MCP enters log operational information into a DISK file with the TITLE *SYSTEM/SUMLOG . Each entry into the file is uniquely identified by the STACKNUMBER of the JOBCODE stack. At the end of the JOB , the entries in the file are edited by the LOGANALYZER and a JOB SUMMARY is printed via a LINEPRINTER. (See Ref. 9 page 6-22 to 6-29).

In order to obtain on-line, the information from the *SYSTEM/SUMLOG file, the relevant entries must be found, edited and sent to the designer.

6.01

Since the correct entries in the *SYSTEM/SUMLOG file can only be found by the STACKNUMBER of the JOBCODE activated as an independent process, the CONTROL ENGINEERING PROGRAMMING PACKAGE must be activated via a WFL JOB and activated as a independent process. All log-operational information can then be received on-line. If the JOBCODE is activated as a dependent asynchronous process then no on-line information is extractable from the *SYSTEM/SUMLOG file but a JOB SUMMARY is available at the end of the session.

The contents of the *SYSTEM/SUMLOG file are,as already mentioned, filled by the MCP and contain the following information.

a) The beginning (BOJ) and the end (EOJ) of the JOB-name and statistics
b) The beginning (BOT) and the end (EOT) of all tasks within the JOB.
c) Reasons for the abnormal ending of tasks
d) If available a STACKHISTORY
e) End of task statistics such as processor time, I/O time and elapsed time
f) Hardcopy information such as- Number of lines printed
Number of cards read or punched
g) File open and close information
h) Messages from Library Maintenance - files copied
- files not copied and reasons

SYSTEM/CEPP is designed to filter out only abnormal conditions when a normal user has activated it. Whenever a special user activates SYSTEM/CEPP options can be set to watch any type of the above mentioned activities. It is then used as an on-line debugging facility.

The main operations performed by SYSTEM/CEPP can be divided into the following phases.

a) <u>The INITIALIZATION</u>

1) The initialization of the I/O files of SYSTEM/CEPP
2) The formation of the unique prefix (CEPP < 4 digits of STACKNO >) for the CEPP CODE files and the user temporary data files .
3) The formation of the unique TITLE of the JOBCODE (CEPP < 4 digits of STACKNO > / SYSTEM .

6.01

b) USER IDENTIFICATION and VERIFICATION (Boolean procedure VALIDUSER)

1) Request user to enter USERCODE and PASSWORD
2) Verify if user is authorized to use the PACKAGE
3) Verify if the USERCODE/PASSWORD combination is valid

c) USER OPTIONS

1) Request if user desires an independent or dependent asychronous process i.e. a separate JOB or not

2) If an independent asynchronous process is desired and the user is a special user then let this user set options to watch certain activities. The default option setting is - All abnormal EOT and EOJ, FILE.. .. NOT COPIED and NO FILE .... .

d) PREPARATION and CREATION of the CEPP JOBCODE

1) Fill the pseudo-card deck with additional information. This information is the designer usercode and password, the CEPP usercode and password, the unique prefix for file TITLEs , information on how the PACKAGE will be activated and the unique JOB name . (procedure FILLPSEUDOCARDS)

2) Convert the pseudo-card deck to the correct WFL MESSAGE format . (procedure FILLWFLMESSAGE)

3) Create the CEPP JOBCODE using the bound-in DCALGOL procedure CONTROLCARDS. Procedure CONTROLCARDS activates the SYSTEM intrinsic procedure WFL-COMPILER .

e) MONITORING of the CEPP JOB

1) If the CEPP JOB has been activated as an independent asynchronous process and the beginning (BOJ) of the JOB can be found by procedure FINDJOB then the messages (edited copies from file *SYSTEM/SUMLOG) to the user will be determined by the watch options.

2) If a dependent process, then SYSTEM/CEPP will wait until the JOB has terminated. Diagnostic messages will be given if errors occur.

6.01

Summing up- SYSTEM/CEPP can initiate the CONTROL ENGINEERING PROGRAMMING PACKAGE as an independent asynchronous task in the following cases.

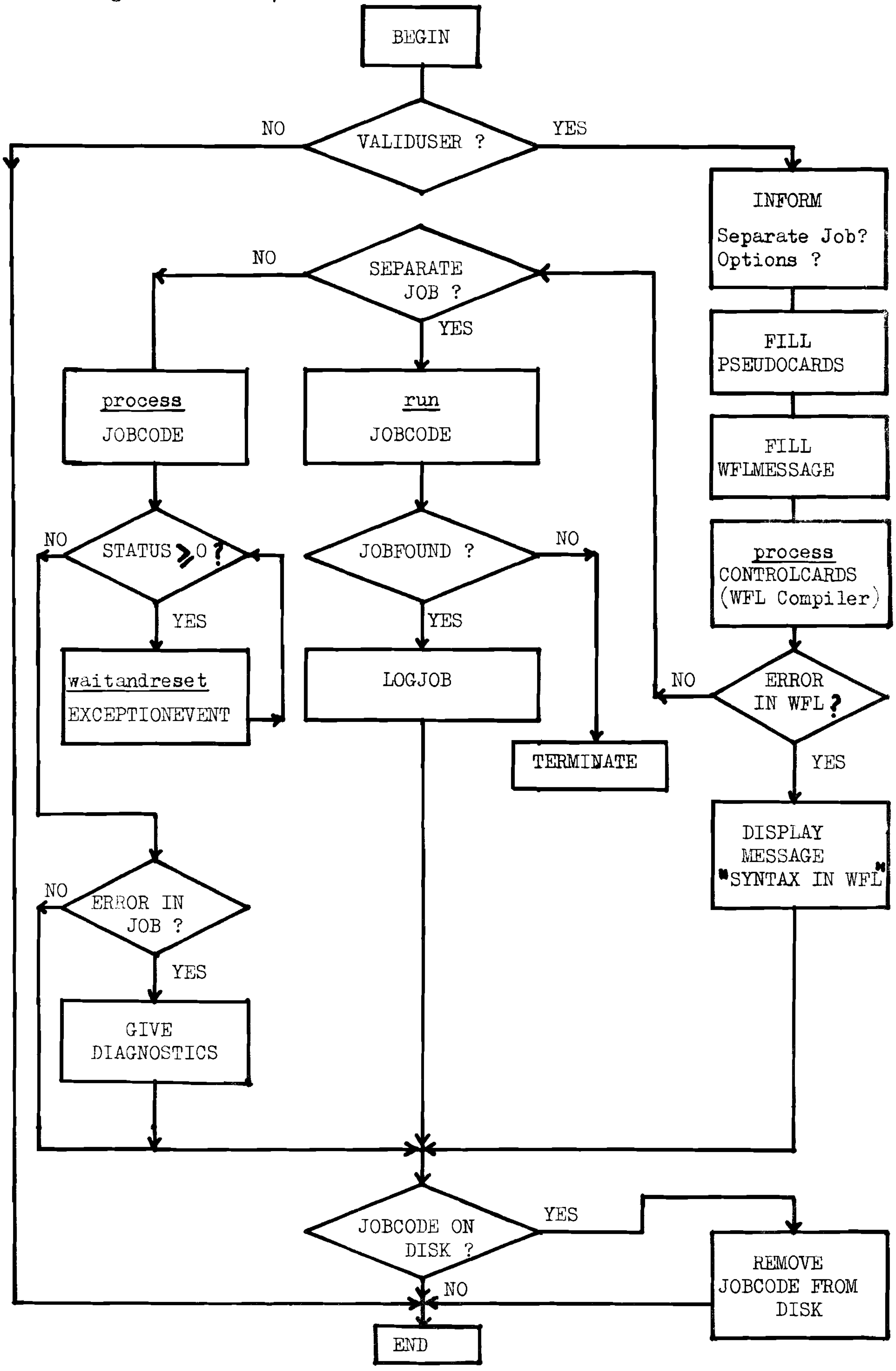
a) Whenever a separate JOB is desired. Under SYSTEM/CANDE the beginning of the SESSION is also the beginning of the JOB. That is to say that a CANDE/SESSION is synonymous with a JOB. A separate JOB will have its own JOB SUMMARY whereas if the CEPP JOB is initiated as a dependent asynchronous task , the entries from the *SYSTEM/SUMLOG file will be added to the CANDE JOB SUMMARY.

b) To provide a more comprehensive surveillance of the PACKAGE . Not all error conditions are visible or accessible programatically . This is specially the case with LIBRARY MAINTENANCE activities .

c) To provide on-line debugging facilities in the case of new program additions etc.

The following page contains a blockdiagram of SYSTEM/CEPP and is intended to give the reader a visual support to the above passages.

Block Diagram of SYSTEM/CEPP



6.02 The SUPERVISOR

The SUPERVISOR is the main program of the CEPP configuration and as such is responsible for the proper functioning of the design programs and its auxiliary program the INPUTSYSTEM. It should be aware of abnormal conditions and attempt to correct these if possible. The designer will recieve appropriate diagnostic messages if errors do occur.

The functions performed by the SUPERVISOR are briefly as follows-

a) The activation of the INPUTSYSTEM and design programs.

b) The monitoring of the INPUTSYSTEM and the design programs.

c) The co-ordination of the access to the REMOTE input file between the INPUTSYSTEM and an active design program .

d) The supplying of suitable diagnostic messages in the case of errors in either the INPUTSYSTEM or the design programs.

The implementation of these functions is shown graphically in the block-diagram of the SUPERVISOR on page 6-9 of this section. The salient features in the block diagram are the two software interrupts HANDLECONTROL and HANDLEIOBREAK and the two COMPLEX WAIT s. The pragmatics of these software constructs have been handled in sections 3.03 and 4.01 .

The interrupt HANDLECONTROL handles all operations resulting from a CONTROL command instruction (See Appendix A) . Such a command may be $GO, $STOP or $BYE . The action taken by the interrupt will depend on the instruction and whether a design program is active or not. The interrupt HANDLEIOBREAK handles all operations resulting from the malfunctioning of the INPUT-SYSTEM . The INPUTSYSTEM must always be restarted if errors occur because it provides the only access to the CEPP configuration. If the INPUTSYSTEM terminates abnormally it will be restarted with a maximum of three times and always restarted in the case of accidental operator intervention .

The two COMPLEX WAIT s can be described briefly as follows-

a) COMPLEX WAIT 1

The SUPERVISOR waits here if no design program is active but can be expected. The STATUS of the design program is therefore -1 .

6.02

b) <u>COMPLEX WAIT 2</u>

The SUPERVISOR waits here whenever a design program is active or has been suspended. If the design program is terminated by the designer disabled or terminated normally the SUPERVISOR will return to COMPLEX WAIT 1 via the DIAGNOSTIC PHASE .

A more detailed account on the workings of the software interrupts will be given at the end of the section.

The following paragraphs will give a brief description of the operations performed by the various phases as portrayed in the block diagram on page .

1) <u>The INITIALIZATION</u>

a) The initialization of the INPUT-OUTPUT files
b) The transference of information from SYSTEM/CEPP which has been passed as parameter. This information is the designer USERCODE/ PASSWORD combination and the unique file prefix-CEPP < 4 digits > .

2) <u>The COMPLEX WAIT 1</u>

a) Wait for a signal from the INPUTSYSTEM that all the necessary items for the activation of a design program are present.

b) While in the WAIT monitor the STATUS of the INPUTSYSTEM (interrupt HANDLEIOBREAKDOWN) .

c) While in the WAIT remain accessible for CONTROL commands (interrupt HANDLECONTROL) .

3) <u>The DESIGN PROGRAM PHASE</u>

This phase is entered whenever the INPUTSYSTEM has notified the SUPERVISOR that a new design program is ready for activation .

a) Copy the contents of the arrays containing the information on the requested design program to arrays not held mutually by the SUPERVISOR and the INPUTSYSTEM.

b) Fill the pointer task attribute NAME and FILECARDS .

c) Activate the design program as a dependent asynchronous process. Signal error conditions if they occur.

6.02

4) The COMPLEX WAIT 2

a) The SUPERVISOR waits here until the design program has terminated normally or abnormally.

b) Remains in this wait if the design program is suspended.

c) While waiting the SUPERVISOR will monitor the STATUS of the INPUTSYSTEM.

d) While waiting it will remain accessible to CONTROL commands from the active design program or from the INPUTSYSTEM if the design program has been suspended .

5) The DIAGNOSTIC PHASE

a) If the design program has terminated abnormally then suitable diagnostic message will be given.

b) If the design program has terminated normally then only the statistics of the design program will be given . (Processor time, I/O time and elapsed time) .

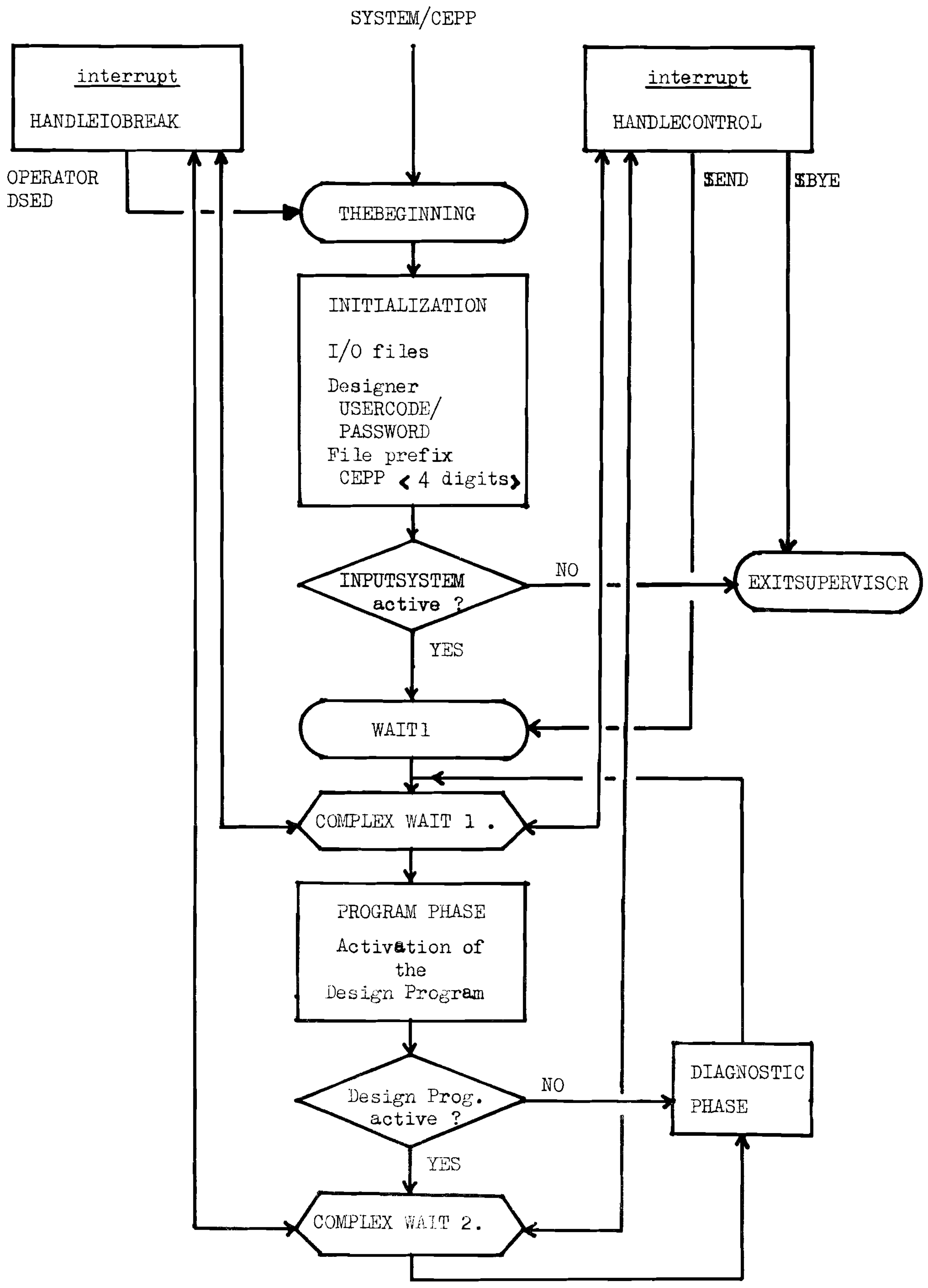
6) The return to the COMPLEX WAIT 1 via a **while** **true** loop

The above is a brief sketch of the implemented algorithm of the SUPERVISOR. The SUPERVISOR is normally in the state of no-operation i.e. in a COMPLEX WAIT . The SUPERVISOR becomes active in the following cases.

a) To activate a design program
b) If errorconditions arise in either the design program or the INPUTSYSTEM .
c) If CONTROL command instructions are entered by the designer .
d) Whenever a design program has terminated .

6.02

Block Diagram of the SUPERVISOR



6.02

In the following paragraphs a slightly more detailed account of the two software interrupts HANDLEIOBREAK and HANDLECONTRL will be given.

a) Interrupt HANDLEIOBREAK

As already mentioned this interrupt is intended to ensure that the INPUTSYSTEM will at all times remain active. This is necessary because all designer instructions are received and interpreted by the INPUTSYSTEM. A special case arises if a design program is also active . If the designer enters an instruction via a design program then the INPUTSYSTEM is activated to process the instruction.
The interrupt is attached to the EXCEPTIONEVENT of the EXCEPTIONTASK of the SUPERVISOR. The INPUTSYSTEM is assigned to be the EXCEPTIONTASK of the SUPERVISOR . The interrupt is executed whenever the value of the task attribute STATUS undergoes a change in value,which will result in the causing of the EXCEPTIONEVENT. The task attribute RESTART of the INPUTSYSTEM has been set to the value 3 ; therefore the INPUTSYSTEM will restart three times before the interrupt is executed by to a program error. If the INPUTSYSTEM is disabled then suitable diagnostics will be given. In the case that the INPUTSYSTEM is disabled by the OPERATOR (by accident or intentionally-due to a NO FILE message) , the OPERATOR is warned the the INPUTSYSTEM will restart and a branch is made to label THEBEGINNING (See Blockdiagram).

b) Interrupt HANDLECONTROL

This interrupt handles instructions of the type CONTROL COMMAND . Such a command can either be given by the designer via the INPUTSYSTEM or via an active design program. The interrupt will determine from which process the instruction came and act on the basis of the instruction. In some cases the INPUTSYSTEM will be reactivated to handle those instructions that pertain to the INPUTSYSTEM.

The analysing of the instruction is done by the MASKSEARCH System Intrinsic and the action to be taken by a case of statement. The MASKSEARCH compares the characters (max. 6) in an array element with the fixed characters in an alpha value array . The value returned depends on the position of the array element of the alpha value array containing the identical character sequence.

6.02

If an identical character sequence can not be found among the given elements of the alpha value array , then the MASKSEARCH will return the value -1.

The value returned by the MASKSEARCH is then used in a case of statement. This case statement contains the instructions to handle the different valid CONTROL COMMAND instructions .

After the execution of the interrupt , the next program instruction performed by the SUPERVISOR will depend on the CONTROL COMMAND instruction. If the designer wishes to terminate the present active design program (CONTROL COMMAND SEND) then the SUPERVISOR will branch to the label WAIT1 . (See the Blockdiagram on page) . In general however , the SUPERVISOR will return to the COMPLEX WAIT where it was before the interrupt occurred.

6.03 The INPUTSYSTEM

The immediate purpose of the INPUTSYSTEM is to act as an interface between the designer and the SUPERVISOR. The instructions from the designer are analysed and converted into BEA software entities . All activities leading to the activation of a design program are performed by the INPUTSYSTEM . It also handles those activities which have no direct bearing on a possible activation of a design program. These activities can be described as utility-oriented i.e. as an aid to the designer.

The activities performed can be divided into the following categories.

a) The interpretation of instructions from the designer. The three types of instructions, the PROGRAM request, the UTILITY request and the CONTROL command are differentiated, analysed syntactically and converted to BEA software entities.

b) The preparation for the activation of a design program. The presence of the to be activated design program code file and the designer indicated source data set files are checked. If necessary these files are copied from storage i.e. from removable disk (DISKPACK) to fixed head-per-track disk (DISK) .

c) The furnishing of utilities for the designer. These utilities can be seen as aids for the designer . These aids take the form of supplying the time of day, the available program names and the names of the data sets created by the designer etc.

The three basic program units a)procedure UTILITYREQUEST
b)procedure PROGRAMREQUEST
c)interrupt HANDLECONTROL

as portrayed in the block diagram of the INPUTSYSTEM on page , clearly reflect the implementation of the three types of instructions. The procedure DATACOM , activated as a dependent asynchronous process , reads messages from the REMOTE input file and does a preliminary scanning of the input to ascertain which type of instruction has been entered by the designer. The COMPLEX WAIT construct is similar to the COMPLEX WAIT encounter in the SUPERVISOR (see section 6.02) . The only difference is that the value returned by the wait intrinsic is assigned to the variable COMMANDTYPE . The value of the variable COMMANDTYPE is then used in a case of statement , branching to either procedure PROGRAMREQUEST or procedure UTILITYREQUEST . The COMPLEX WAIT and software interrupt structure and their functioning is identical to the similar construct described in section 6.02 on the SUPERVISOR.

The following paragraphs will contain descriptions of the basic program units as portrayed in the block diagram of the INPUTSYSTEM on page .

a) The INITIALIZATION

This program segment attaches and enables the software interrupt HANDLECONTROL . Interrupt HANDLECONTROL is attached to event CONTROL. As a service to the designer , the time of day, month and year are given. The important operation performed is the determination of the availability of the design programs. The directories of a number of removable disk (DISK-PACK) storage devices are searched for the presence of a standard program TITLE . If this TITLE is found , it is assumed that the other (if any) are also present on that particular storage device . If it is not found then the designer will be notified and the session will be aborted.

b) The PRELIMINARIES

The preliminaries consist of two procedures - GETUSERNAME and USERTEXT . Procedure GETUSERNAME asks the designer to enter his name . The name given is used as an identifier in the TITLEs of the designer created or to be created data sets . In this way , the names of the data sets are unique although two designer could be working simultaneously under the same USERCODE/PASSWORD combination. Procedure USERTEXT prints out an abridged version of Appendix A , instructing the designer how to formulate his instructions. The preliminaries are bypassed if the INPUT-SYSTEM has restarted due to a program error .

c) The INPUT Process

Procedure DATACOM performs as primary input procedure for the CEPP structure. The REMOTE input file is switched between procedure DATACOM of the INPUT-SYSTEM and the INPUT procedure of an active design program.

6.03

c) Cont.

Procedure DATACOM handles the following operations.

1) The scanning of the input string from the REMOTE file for illegal characters . If there are illegal characters then the input string is rejected and the offending character is indicated .

2) Determines the type of instruction entered by the designer. If the first character if the instruction string is -
"$" a Dollar sign then it is assumed to be a CONTROL command
"#" a Crosshatch then it is assumed to be a UTILITY request
otherwise it is assumed to be a PROGRAM request if the first character of the string is an alpha character.

3) On the basis of the type of instruction entered by the designer the following events are cause'd .
If a PROGRAM request then event REQUEST
If a UTILITY request then event UTILITY
If a CONTROL command then event CONTROL

Because procedure DATACOM is an asynchronous process , it can be active simultaneously with the other parts of the INPUTSYSTEM. Safeguards have been implemented to ensure the correct handling sequence of the incomming instructions. If the previous instruction has not been completed then the instruction will be rejected except in the special case of the CONTROL command . A CONTROL command will always be accepted if the(possible) previous CONTROL command has been completed. The particular CONTROL instruction $BYE will result in the termination of the process DATACOM and in turn the termination of the other active processes .

4) The switching of access to the REMOTE input file. The construct used is as follows. If a design program has been successfully activated by the SUPERVISOR, the design program will wait until the designer has entered the CONTROL command $GO,before resuming processing. The STATUS task attribute of the design program is visible to the procedure DATACOM and is checked , along with the CONTROL command instruction.

6.03

If the two conditions are met viz the STATUS of the design program equal to 2 and the instruction is $GO then a waitandreset is entered instead of the READ from the REMOTE file. The process DATACOM will wait until the event INTERCOM is cause'd by the SUPERVISOR . The SUPERVISOR cause's the event INTERCOM whenever the designer program is terminated (normal or abnormal) or in the case of a temporary suspension of the design program on request from the designer. The causing of the event INTERCOM will result in the exiting of the waitandreset and the entering of the READ statement . In the mean time the design program is in a waitandreset so that the procedure DATACOM is the only process that has access to the REMOTE file .

d) The PROGRAM Request

The procedure PROGRAMREQUEST performs all the preparation necessary for the successful activation of a design program. The syntax of the PROGRAM request instruction is handled in Appendix A . The operations performed by this procedure will be discussed on the basis of the syntax of the PROGRAM request instruction, the general format of which is -

< destination identifier > := < program identifier > [< source list >]

The < source list > is a number of source identifiers delimited by commas.

The operations performed can be ennumerated as follows -

1) The instruction is checked on syntax and error messages to the designer are formulated if errors occur.

2) If the input string is syntactically correct then the individual items are copied to fixed arrays. These items are the source identifier, the program identifier and the three possible source identifiers. (The above two operations are performed by procedure USERREQUEST)

3) The program indicated by the designer is then verified . The code file for the program could be present on the fixed head-per-track disk or on removable disk (DISKPACK) . If the code file is on DISKPACK then it must be copied to the fixed head-per-track disk . In the present implementation of the INPUTSYSTEM this accomplished in a roundabout way which will be described in the paragraph on the copying process .

6.03

d)3 Cont.

If the design program code file is not present on the fixed head-per-track disk or on the removable disk then the designer will be notified and further verification will be aborted.

4) The next items to be verified are the designer given source identifiers. Each source identifier is associated with a particular data file of the designer. A designer data file may be present on DISK (if created during the present session) or on DISKPACK . If the data file is not present on DISK then the directories of the removable disks are searched . If the data set can not be found then the designer is notified and the rest of the verification is aborted. The information on the whereabouts of the designer data set (s) is stored in arrays to be passed on to the design program by the SUPERVISOR .

5) The last item to be verified is the destination identifier . The designer could by accident or intentionally use a destination identifier of a data set that is already existent on DISK . If this is the case the designer is notified and ask to explicitly state his intention.

All the prerequisites for the successful activation of a design program have now been verified (pending the successful copying of the code file) . The SUPERVISOR can be notified that it can activate a design program. This is done by <u>cause</u>'ing the <u>event</u> NEW10 which will result in the SUPERVISOR entering the PROGRAM PHASE (See section 6.02) . If the designer attempts to activate a new design program before a present active design program has terminated,the designer will be notified and the PROGRAM request will be rejected.

e) <u>The Copying Process</u>

In the present implementation the copying process is handled in very roundabout fashion compared to the copying operations performed by the program SYSTEM/CEPP in section 6.01 . The copying of files is achieved by creating a DISK file containing the WFL COPY statements . An auxiliary program called SYSTEM/WFL is then activated and the DISK file label equated. SYSTEM/WFL in turn activates the WFLCOMPILER and creates a

6.03

e) Cont.

a JOBCODE file of the WFL COPY statements. This JOBCODE file is then activated as a dependent synchronous process. It goes without saying that this COPY process is in need of updating . The method employed in the program SYSTEM/CEPP is directer and more efficient , (by activating the WFLCOMPILER by a DCALGOL procedure) .

f) The UTILITIES

A utility instruction is an instruction prefixed by the crosshatch symbol. They are intended to aid the designer in the design process. All utility instructions are handled by the procedure UTLITYREQUEST . The interpretation of the instructions is done with the usual MASKSEARCH of an alpha value array containing the valid instructions. The value returned by the MASKSEARCH intrinsic is then used to branch the corresponding set of instructions via a case of statement. Measures have been implemented to prevent the designer from entering new instructions before procedure UTILITYREQUEST has terminated . This is done by procedure DATACOM which will reject any instruction that is not a CONTROL command until procedure UTILITYREQUEST has finished processing.

g) The Software Interrupt HANDLECONTROL

The interrupt HANDLECONTROL handles all CONTROL commands. These CONTROL commands can arise from the following sources-

a) From procedure DATACOM as entered by the designer

b) From the design program as entered by the designer

c) Created programmatically by the SUPERVISOR or by the INPUTSYSTEM itself.

The interrupt can be attached to either event CONTROL or event IOCONTROL depending the accessibility of the REMOTE input file . If no design program ia active then the interrupt is attached to event CONTROL . It is attached to event IOCONTROL whenever a design program is active but not suspended temporarily. If the design program is suspended temporarily it is in a waitandreset which is not the same as suspended when the STATUS is equal to 3 .

6.03

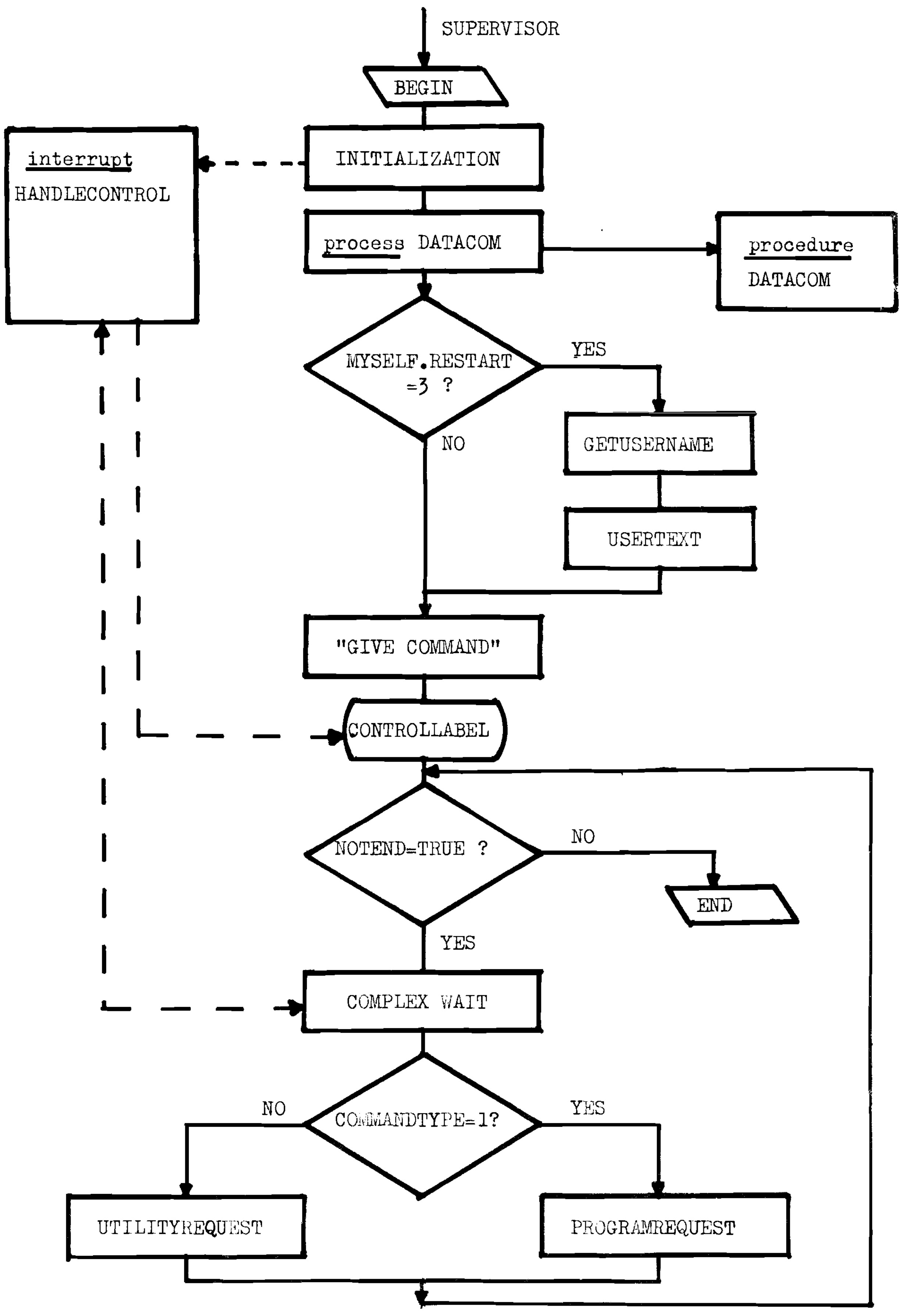
g) Cont.

Whenever the designer enters the instruction $GO the interrupt is executed. The STATUS of the design program must be equal 2 otherwise the command is rejected. The interrupt then attaches itself to event IOCONTROL and returns to the COMPLEX WAIT . The INPUTSYSTEM is reactivated by the SUPERVISOR if the event IOCONTROL is cause'd . The interrupt also contains the necessary checks in order to ensure a correct sequence of possible CONTROL commands. For instance if the designer enters the command $STOP but no design program is active then the designer will be notified and the instruction rejected.

Another point of interest is the fact that there is form of communication between the interrupt HANDLECONTROL of the SUPERVISOR and the interrupt HANDLECONTROL of the INPUTSYSTEM . The events SUPCONTROL and IOCONTROL are used to coordinate the activities between the two interrupts whenever necessary . If , for example the designer has entered the command $END with the intention of terminating the present active design program (this command is only accepted after the command $STOP has been entered) then the SUPERVISOR must terminate the design program. Once the design program has terminated , the SUPERVISOR will cause the event IOCONTROL . The interrupt has been waiting for this event to happen . If the event IOCONTROL has been caused then the design program has terminated and therefore the designer can enter a new command.

In the above passages the principle program units of the INPUTSYSTEM have been described in their functions,and the operations performed to fulfill those functions. Some improvements can be made (i.e. the Copy Process) to increase the efficiency and dependability of the program. New instructions can be incorporated with a minimum of difficulty with regard to the UTILITY request instruction . The CONTROL commands implemented should provide the designer with sufficient control over the activities of the design program. The major feature of the implementation is the extensive use of variables of the type event . These variables are used to coordinate the activities of processes and to provide the necessary No-Operation state by using the wait system intrinsic.

6.03 Block Diagram of the INPUTSYSTEM



6.04 The DESIGN Programs

The operation performed by a design program has been envisaged as a transformation or a mapping of one data set into another (See Section) . Such a transformation can be broken down into the following basic activities-

a) The INPUT PHASE

b) The COMPUTATION PHASE

c) The OUTPUT PHASE .

These basic activities are of course also performed in normal programs . In most cases however the activities are intermixed ; for example part of the input data is read, certain computations are performed and the results sent to the LINEPRINTER (say) , then further input data is read and so forth .. The numeric input information is supplied generally speaking via a CARDREADER file or a disk file of the type DISK or DISKPACK. The programmer knows how the data is structured in the disk files and will program his read statements accordingly . If the input is to be read from a CARDREADER file then the programmer can either structure his card deck according to the read statements in the program or change the read statements. The results of the computation are,in most cases,sent to the LINEPRINTER only . Whenever all or part of the output is stored on a disk file , it is generally intended that this data is to be read by a particular program in a particular way. This means however that the program producing the output and the program using this output as input will be inter-dependent . Any changes in the OUTPUT PHASE of the first program will necessitate the alteration of the second program. In the case of the design process,where any number of design programs produce output to be used again as input data , this is a highly undesirable state of affairs.

What is desired of a design process can be put as follows-

a) That each design program can extract the information it needs for the computation from a given data set.

b) That each design program produces an ouput data set in such a way that another design program can perform (a) .

If each design program can perform these two operations then it becomes to a large extent independent of other design programs. The program then becomes self-supporting in the sense that it will be indifferent to the data set assigned to it by the designer. If the program can extract all the information it needs for the computation , it implicitly accepts the designer assigned source data set as being a valid data set. Otherwise it will reject the data set or demand further information from the designer. The implementation of such a scheme will result in the much desired property of "modularity" . The set of available design programs will then consist of a number of modules. An existing module can be omitted or a new module added without affecting the other modules. An extra bonus derives from the fact that the designer will not need to know which data set can be used with which program. The design program will simply notify the designer if it can not use the assigned data set. The advantages accrued by the modular structure of the design programs will have to be paid for by extra software support in each design program and a more complex structure of the data set files.

Besides performing the operations directly related to the mapping process , each design program should also be able to-

a) Catch run-time arithmetic errors during the computation

b) Remain accessible for designer entered CONTROL commands .

The handling of run-time errors can be accomplished by the BEA ON FAULT statement or by the SUPERVISOR . If handled by the SUPERVISOR , the design program must be reactivated by a PROGRAM request instruction .

The design program will remain accessible for CONTROL commands if the REMOTE input file is read by an asynchronous dependent process activated by the design program. This process can be made to scan for valid CONTROL commands entered by the designer and if necessary activate a software interrupt. During important phases of the design program the software interrupt can be detached, so that the possible return from the interrupt to the last-but-one performed instruction.

Appendix A

Provisional Users Guide

The Control Engineering Programming Package is implemented with three types of instructions-

a) The PROGRAM request
b) The UTILITY request
c) The CONTROL command

In short, a PROGRAM request instruction is a request for a specific design program such as NYQUIST or ROOTLOCUS, a UTILITY request is a request for some form of aid such as the present status of a design program and a CONTROL command is an instruction for the commencing, suspending or terminating of a design program. The following will be a more detailed account of each of the three types of instructions.

a) The PROGRAM Request

The syntax of the PROGRAM request instruction , using the Backus-Naur notation with the metalinguistic symbols . . , ::= , | can be described as follows. (For a short description of the meanings of these symbols see Ref. 10)

```
< program request >  ::= < destination identifier >  := < program identifier >
                                   [ < source list > ]   |
                         < program identifier > [ < source list > ]

< source list >      ::= < source list > | < source list > , < source
                                                              identifier >

< source identifier> ::= < BEA identifier > | < CEPP identifier > |  *

< destination ident. >::= < BEA identifier > | < CEPP identifier >

< program ident.  >   ::= < BEA identifier >

< BEA identifier   >  ::= < letter > | < BEA identifier > < letter > |
                                     < BEA identifier > < digit  >
```

```
‹ CEPP identifier › ::= ‹ letter › | ‹ CEPP identifier ›‹ letter › |
                        ‹ CEPP identifier › ‹ digit › |
                        ‹ CEPP identifier › ‹ special character ›

‹ special character › ::=  ( | )
```

N.B. A maximum of 3 source identifiers is permitted in the source list.

The symbol * , the asterisk as a source identifier is used when no data file is available or is about to be created. The following will be an example of valid program requests given in a meaningful sequence.

<u>Instruction Number</u>	
1	B(S):= POLY[*]
2	NYQ1:= NYQUIST[B(S)]
3	PLOT[NYQ1]

Instruction nr. 1 means that the user desires to create a data set of a transfer function in polynomial form. The data set will be named B(S) . The program POLY will ask the user to enter his data in a particular sequence. Once the data set is filled the user can enter the next instruction.

Instruction nr. 2 performs the mathematical operation called NYQUIST on the data set called B(S) and creates a new data set called NYQ1. In this case the new data set NYQ1 will contain the real and imaginary parts of of the transfer function B(S) for a certain frequency range.

Instruction nr. 3 initiates the program PLOT . The program PLOT will use the data set NYQ1 to produce a NYQUIST diagram on a given(in program PLOT) plotter device.

Other programs may use the same data set. Each program prepares its output data set for its successor(s) . Instruction nr. 3 could have been -WRITE NYQ1 . In this case the data set NYQ1 would then have been printed out on the lineprinter.

Summerizing, the PROGRAM request construct permits the user-

a) to name his own data sets with BEA or CEPP identifiers

b) to indicate which data sets are to be use as source for the design program

c) to name the resultant data set

d) to initiate any available CEPP program

b) <u>The UTILITY Request</u>

The utility request is designed to give the user supplementary information. All utility requests are prefixed by the symbol "#" (crosshatch) . The following utility requests have been implemented.

1.	# PROGRAMS	- gives the user the names of available CEPP programs.
2.	# TIME	- gives time of day,day of the month,month of the year and the year.
3.	# STATUS	-gives the present state of a CEPP program If the program is active or has terminated the elapsed,process and I/O time will be given. If the program is waiting to be activated the user will be given notice to that effect.
4.	# DATA	-(to be implemented) will give the user the names of his data sets.

c) The CONTROL Command

The CONTROL command is designed to give the user some measure of control over the progress of his program(s) .The user should be able to stop the processing at will. He could then ask for information via a UTILITY request and either resume processing or terminate the program.
All CONTROL commands are prefixed by the symbol "$" (dollar sign) .

The following CONTROL commands have been implemented.

1.	$GO	-All CEPP programs when active will give notice to the user that it has started.The program will wait until this command is given by the user.
2.	$STOP	-If a program is active i.e. being processed then this command will cause the program to be interrupted and suspended. It will wait until the user gives either a $GO command or $END command
3.	$END	-If a program is active this command will cause the program to terminate before the normal end of the program.
4.	$BYE	-This command causes the termination of the CEPP user session. All active programs are terminated.

If the command $STOP is given then only a UTILITY request or a CONTROL command can be given. Any attempt to start up a new program via a PROGRAM request will be discarded. Any active program must terminate normally or be terminated by the $END command before a new PROGRAM request is accepted.

Appendix B

Binding Fortran Programs to Algol Procedures

It is possible to bind Fortran programs into Algol programs using the Algol Compiler Option - $ SET AUTOBIND . If this option is set in an Algol program then all missing code segments(separately compiled Fortran subroutines) will be automatically bound into the Algol code.

In general a Fortran program will consist of a main program and a number of subroutines. If the main program is also made a subroutine then it can be bound into the Algol program as a procedure .
The general scheme can be outlined as follows-

a) Remove all file declarations from the Fortran main program because the file declarations of the Algol host can be used. This also avoids the complication of Fortran files being global to the Fortran program body. All READ and WRITE statements in Fortran can be left unchanged.

b) Declare the Fortran main program as a subroutine. Include the Fortran Compiler Option - $ SET SEPARATE because Fortran subroutines are not permitted to be compiled without a main program as with an Algol procedure.

c) The original subroutines of the Fortran main program can be left unchanged.

d) Compile the Fortran deck for LIBRARY (i.e. DISK)

e) Declare the Fortran subroutine containing the main Fortran program as an external procedure in the Algol program. Specify which Algol file identifiers the compiler must use for the Fortran files and of course include the compiler option $ SET AUTOBIND .

f) Compile the host Algol program. The AUTOBIND option will cause those procedures which are declared as external and located on DISK to be added to the procedure declaration as being the missing procedure body.

The following pages give a complete example of the binding of a Fortran program with a subroutine to an Algol procedure . The Algol procedure also has a formal parameter . The procedure will receive the actual parameter via a CONTROL card and pass this value to the Fortran subroutines.

The following points are essential in using the Algol Compiler Option
$ SET AUTOBIND.

a) The HOST must be in Algol
b) The Fortran segments must 1) Have the same directory as the Algol HOST
2) Be previously compiled
3) Be located on DISK

The Fortran segments are destroyed in the Binding operation unless the compiler option $ BIND subroutine identifier is used.

If the Algol HOST contains external procedure declarations which are <u>not</u> to be bound in then the compiler option $ EXTERNAL procedure identifier should be inserted.

If the Fortran subroutine contains file identifiers which have not been declared then the declaration in the HOST is used. The Algol compiler will use the HOST file identifier for the Fortran file identifier if the following compiler option is used- $ USE IN FOR FILE3 .
File IN is declared in the Algol HOST and file 3 is the Fortran file.

```
begin
file IN(KIND=READER)                        SUBROUTINE FORT
READ( IN, .... );                           READ( 3,100)
                                       100  FORMAT(... )

                                            RETURN
                                            END
end.
```

The effect is that the Algol file identifier IN is used for the Fortran file identifier FILE3 .

The parameters that can be passed between an Algol HOST and Fortran subroutines are limited to simple variables and one dimension 1 arrays.
N.B. If an Algol array is declared as A 0:99 then the Fortran will be DIMENSION A (100). That is to say the Algol array will begin at A 0 whereas the Fortran array will begin at A (1) .

References

1 Work Flow Management-Reference Manual
Rdok 98 - Burroughs 5000709

2 Work Flow Management -User's Guide
Rdok 99 - Burroughs 5000714

3 Program Binder
Rdok 93 - Burroughs 5000045

4 Using Pointer Expressions on the B6700/B5700 Computing Systems. Rdok 97 - Burroughs 4000095

5 Input/Output Subsystem
Rdok 92 - Burroughs 5000185

6 User's Guide to Memory Control
7 Tasking and Inter-Program Communication
Chapters 1 and 6 of System Miscellanea
Rdok 92 - Burroughs 5000367

8 Inter-Program Communication
The B6700 Hot Line
Rdok 76 - Burroughs 1042298-012

9 System Software Handbook
Rdok 75 - Burroughs 5000276

10 Algol Language Reference Manual
Rdok 107 -Burroughs 5000649

11 Computer System Organization. Elliot I. Organick, The B5700/B6700 Series, Academic Press, 1973 .

12 Virtual Memory. P.J. Denning , ACM Computer Surveys 2(3),153-189, 1970 .

13 The Working Set Model for Program Behaviour. P.J Denning, Communications of the ACM, Vol. 11 , Number 5, May 1968 , 323-333 .

14 On-line Design of Control Systems. N. Munro, The Computer Bulletin, Vol. 14 p 184-186, 1970 .