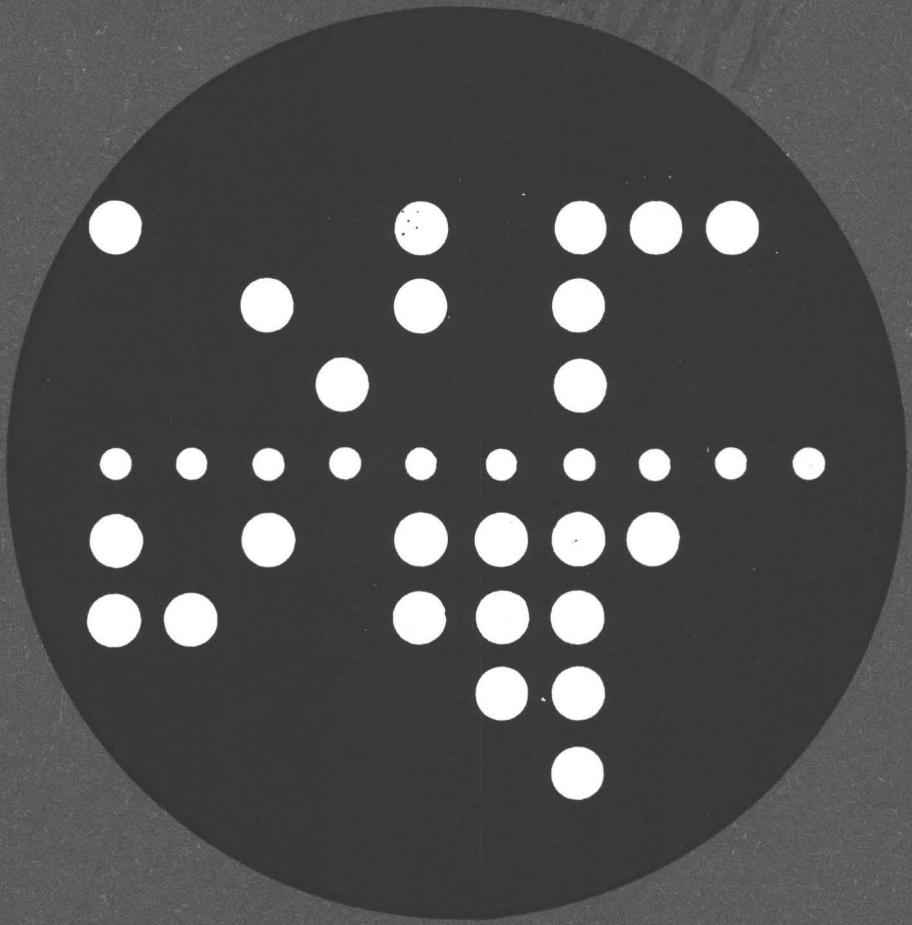
COMPUTING CENTRE NEWSLETTER

JRC - TSO Primer

LIBRARY



Commission of the European Communities
JOINT RESEARCH CENTRE
Ispra Establishment

SPECIAL ISSUE

JRC - TSO Primer

A First Introduction to the Use of the TSO System
as Installed at the JRC Computing Centre, Ispra.

A. Rink

August 1979.

CONTENTS

1. Introduction

The aim of this paper is to give a first introduction into the usage of some basic facilities of the Time Sharing Option (TSO) of the IBM 370 Operating System, which is available at the Ispra JRC Computing Centre.
As a second aim the paper intends to provide some basic knowledge on operating systems, since this might be helpful in understanding certain properties and terms of TSO.

The purpose of an operating system is to enable a group of people to share a computer installation efficiently. This means that people will compete for the use of physical resources such as processor time, storage space, and peripheral devices (card readers, printers, etc.). The sharing of a computer installation is an economic necessity [1]. To allow this an operating system must have a policy for choosing the order in which competing users are served and for resolving conflicts of simultaneous requests for the same resources. It also must have means of enforcing this policy in spite of the presence of erroneous or malicious user programs and, since users must pay for the cost of computing, accounting of the usage of resources has to be performed [1].

Operating systems can be distinguished by the way users can request work to be done.

In the following, two types of operating systems will be considered (batch-processing systems and interactive systems) and their advantages and disadvantages will be discussed.

Finally TSO which belongs to the class of interactive systems will be presented and its use will be shown in connection with example sessions.

2. Batch-processing systems

A computation requested by a user is called a job. A job may consist of several separate programs to be executed sequentially, each individual program being called a job step. A job is defined by using a job control language.

```
THIS IS A JOB DEFINITION.
I'M USER MICKY MOUSE AND
MY ACCOUNT NUMBER IS 111.
AS FIRST STEP THE FOLLOWING
PL/1-PROGRAM HAS TO BE COMPILED.
PL/1-PROGRAM :  PROC OPTIONS (MAIN);
     .
     .
     .
END PL/1-PROGRAM;
AS SECOND STEP THE COMPILED PROGRAM
HAS TO BE EXECUTED.
END OF JOB DEFINITION.
```

Fig.2.1 Example of a job definition

Fig. 2.1 shows an example of a job definition. For didactic reasons a natural language (English) was used as job control language. However, in real life this is not the case.

Typically in batch-processing systems job definitions are punched on cards and submitted to the system via card readers. Figure 2.2 shows the basic organization of such a system [1]. (In [1] a distinction is made between batch-processing systems and spooling systems which will not be used here since from the user's point of view both system types are considered as non-interactive).

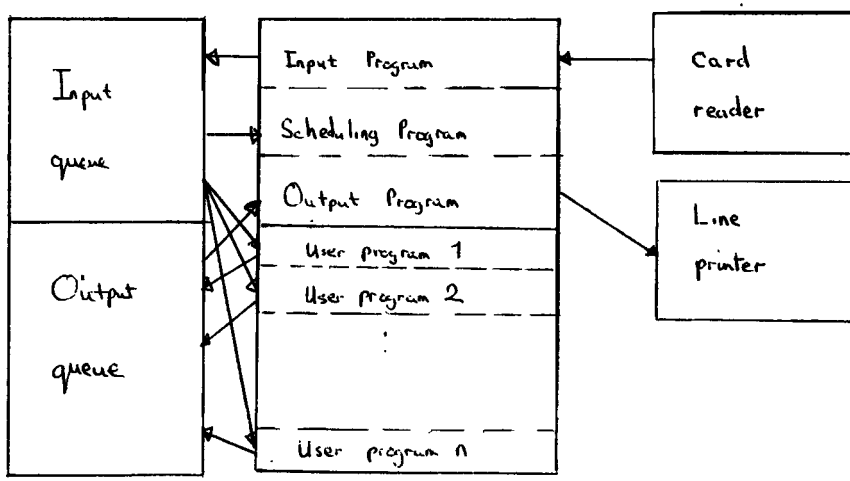


Fig. 2.2 Organization of a batch-processing system.

The central processor is multiplexed between several programs. That is, program execution is interleaved in time. The input program reads cards from the card reader to a queue on the backing store (disk, drum). The scheduling program selects user jobs from this input queue and starts their execution. The output program prints output read from the output queue on a line printer. The user programs held in the internal store read their data from the input queue and write results in an output queue on the backing store.
The operating system consists of the input program, output program and scheduling program. The number of user programs which can be executed concurrently depends on the availibility of the internal store. If there are very large programs stored on backing store only few of them can run concurrently.
Operating systems like the one described tend to be effective with respect to throughput (average number of jobs executed per time unit) and processor utilization.

3. Interactive Systems

To improve interaction between a computer and human beings, users should be given the possibility to use the facilities of a computer interactivly.
This means, that users can define their requests in an uncomplicated way and the system responds to trivial requests (Like data editing requests) within a few seconds. So what is wanted is to have a computer made simultaneously avilable to many users in a manner somewhat like a telephone exchange. Each user would be able to use a typewriter like device at his own pace and without concern for the activity of others using the system.
Operating systems which possess these features are called interactive systems. Fig. 3.1 shows the basic organization of an interactive system [1,2].

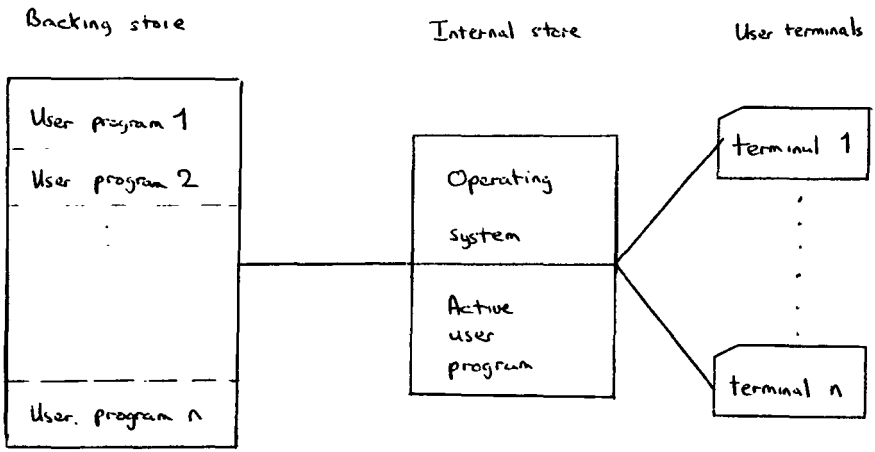


Fig. 3.1 Organization of an interactive system.

The internal store is divided into two parts, one for the operating system, the other part for the user programs. A large backing store (drum or disk) is divided into n slices, where each slice is of the same size as the user area in the main store. Programs and data of a user reside either in the user area of the internal store, or in one of the slices on the backing store.

Computation requests and user programs are entered via the terminals.

The operating system takes care of the terminals. Assume that all user programs are requested to run. Since there is only one user area within the internal store the operating system sequentially loads one program, keeps it there for a certain amount of time (for instance 0.1 sec) called a time slice, and assigns to it the processor. After its time slice is expired the user program is stored back on backing store and the next program is loaded. This process is repeated in a cycle until all computation requests are satisfied.

The process of copying user programs back and forth between internal and backing store is called swapping.

The technique of sequentially running each user program for a short amount of time is known as time-sharing. Therefore interactive systems are also referred to as time-sharing systems.

Besides the commands needed normally to request computations the control languages in interactive systems provide a lot of additional facilities to support users in doing their work.

These facilities include commands to enter, modify, store and retrieve data, to develop, test and debug programs, and to control the terminal session.
Fig. 3.2 shows an example of a terminal session.

```
ENTER LOGON
logon user=micky mouse account nbr = 111
PREPARATION OF TERMINAL SESSION FOR  MICKY MOUSE  IN PROGRESS
READY
edit example.pl1 new
INPUT
program:  proc options (main);
  .
  .
  .
end;
save
SAVED
READY
run example.pl1
READY
logoff
END OF TERMINAL SESSION
```

Fig.3.2 Example of terminal session

In the above example all system messages are written using uppercase letters and all user input is written using lowercase letters.
At the beginning of the terminal session the user is asked to identify himself. After the system has verified the user input it sends the message READY. Now the user creates a program called EXAMPLE.PL1 by using the EDIT command. After saving it on backing store for later use the user requests its compilation and execution with the RUN command. Finally the user finishes his terminal session with LOGOFF.

4. Interactive systems vs batch-processing systems

Based on the system descriptions given in chapter 2 and chapter 3, in this chapter the advantages and disadvantages of interactive systems in comparison with batch-processing systems will be discussed.

1. Advantages

Fast response to trivial requests

In interactive systems the time interval between the request for a computation and the return of its results is called response time. In batch-processing systems this time interval is defined slightly differently and is called turnaround time, that is the elapsed time between the submission of a job to the computer centre for processing and the return of its results to the programmer. In interactive systems the units of interaction are much smaller than in batch-processing systems. Also in interactive systems the basic scheduling policy is to guarantee a fast response to trivial requests while in batch-processing systems the basic scheduling policy aims to have a good throughput. Consequently, response time to trivial requests is measured in seconds while turnaround time is measured in minutes and sometimes in hours.

Improvement of programming and debugging

One feature of interactive systems is that they give a fast response to trivial requests. Thus, they are able to provide a lot of additional facilities to support users in doing their work. These include facilities to enter, store, retrieve, and edit programs from the terminal. For debugging special test facilities are offered to test programs from the terminal. In this way programs can be stopped at predetermined points, the contents of variables can be listed on the terminal, and new values can be set.

Conversational access

Conversational access means that a computer can be accessed directly from a typewriter-like device and that for trivial requests response time is very short (a few seconds).
This type of conversation is much more adapted to human beings. In addition the language which has to be used to express computation requests can be simplified since in cases where the user does not know what to do the system will supply additional information. In cases where the system misses some information it will ask for it.

2. Disadvantages

Low processor utilization

In interactive systems a short guaranteed response time is achieved at the price of decreased processor utilization. This is due to the fact that these systems are forced to lose processor time on unproductive transfers of programs between two levels of store [1].

Slow down of large computations

In interactive systems every user requested computation which needs multiple time slices is slowed down by a certain factor due to the time used for swapping, its involved overhead, and because of the need to serve other users on a 'good' response time basis. So interactive scheduling only makes sense for more or less trivial requests; it is not a realistic method for computations that run for minutes and hours [1].

Lower throughput

Throughput is defined in this document as the average number of jobs executed per time unit.
Because of the high system overhead inherent in interactive systems, throughput here is achieved at a lower rate than in batch-processing systems.

3. Conclusion

As it was mentioned before time-sharing systems and batch-processing systems have different objectives. That is, they serve different purposes.
Interactive system aim to give a fast response to trivial requests of simultaneous users.
Therefore, they are very useful for computations where human beings are involved and where the pace is limited by human thinking.
Batch-processing systems, however, aim to have a good throughput. That means that they are well suited for serious computations where no interactions with human beings occur. Thus, these two system types do not compete with each other. Each gives the users a special service in a very efficient manner.

5. IBM System/370 Operating System: Time Sharing Option (TSO)

5.1 Introduction

TSO is the time sharing system of the IBM System/370 Operating System. TSO allows a number of users to use the facilities of the system concurrently and in a conversational manner. Users can communicate with the system by typing requests for work on a terminal. The system responds to those requests by performing the work and sending messages back to the terminal [4,5].
In TSO computation requests are expressed through commands. By using different commands, different kinds of work are carried out.
When a command is used to request work, the command establishes the scope of the work to the system. For some commands, the scope of the work encompasses several operations that are identified separately. After entering the command, one of the separately identifiable operations (subcommands) may be specified. A subcommand, like a command, is a request for work. However, the work requested by a subcommand is a particular operation within the scope of work established by a command [4,5]. The commands and subcommands recognized by TSO form the TSO command language.
In the following sections the basic facilities of the TSO command language will be explained and examples of its usage will be given. Thereby the following conventions have to be observed:

- any user input is written in lowercase letters.

- all system messages come out in uppercase letter.

- due to didactic reasons all keywords of the command language are underlined.

- keywords have to be separated by one or more spaces (blanks).

- to make the user input available to the system every input has to be finished by pressing the RETURN key of the used terminal. In the following examples it is illustrated by (CR)

- the backspace key can be used to delete the preceding character on an input line.

- the ATTN key can be used to delete the entire input line.

At the end of this chapter two complete example sessions are given.

5.2 Starting and ending a terminal session

To start a terminal session the first thing that has to be done is to turn on power at the terminal and possibly at the modem. After this the LOGON command to identify the user to the system has to be used. Thereby the system needs the following information:

1. User identification - the name or code by which the user is known to the system.

2. Password - A further identification used for additional security protection.

3. Procedure name - The name of a series of statements that predefine a certain kind of work the user wants to do.

For all examples within this chapter the following LOGON information will be used:

- user identification = tsotest
- password = passall
- procedure name = pl1log (this tells the system that PL/1 will be used as programming language)

Example 5.2.1:

```
turn ON/OFF switch to on
a (CR)
IKJ54012A ENTER LOGON -
logon tsotest/passall proc(pl1log) (CR)
TSOTEST LOGON IN PROGRESS AT 10:00:23 ON JULY 7,1978
NO BROADCAST MESSAGES
CPU - 00:00:01 EXECUTION - 00:00:16 SESSION - 00:00:24
READY
```

To end a terminal session the LOGOFF command can be entered.

Example 5.2.2:

```
READY
logoff (CR)
TSOTEST LOGGED OFF TSO AT 10:37:17 ON JULY 7,1978+
turn ON/OFF switch to OFF
```

5.3 The HELP command

The HELP command provides the user with information about all other TSO commands. At the most general level the user can enter :

help (CR)

This will cause the user to receive a list of all commands and a brief explanation of their functions. If the user wants all the information available on a specific command, for example EDIT, the following should be entered:

help edit (CR)

If one wants only the function, syntax, or operands, of the EDIT command, one of the following should be entered:

help edit function (CR)
help edit syntax (CR)
help edit operands (CR)

5.4 Entering and manipulating data

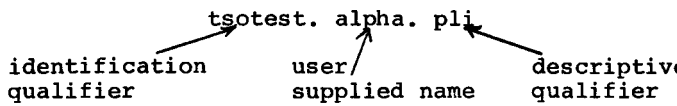
5.4.1 TSO data set naming conventions

Almost all system applications are concerned with the processing of data. Therefore, it is important to know how to enter data into the system and how to modify, store, and retrieve data after it has been entered [4].
Any collection of related data treated as a unit is called a data set. For example, a data set may contain:
- text used for information storage and retrieval,
- a source program,
- data used as input to a program.

To uniquely identify each data set stored in the system it must be given a unique name. The system then uses that name to identify the data set whenever it is to be accessed. In TSO a data set name consists of three fields seperated by a period:

1. The user identification
2. The user supplied name
3. A descriptive qualifier

Example 5.4.1.1:



Each field consists of one through 8 alphanumeric characters and must begin with an alphabetic character.
The identification qualifier is the user identifaction specified within the LOGON command. The descriptive qualifier specifies the type of data contained in the data set. A subset of descriptive qualifiers known to TSO is given below.

Descriptive qualifier	Data set contents
ASM	Assembler program
COBOL	COBOL program
DATA	Uppercase text
FORT	FORTRAN program
LIST	Listings
TEXT	Uppercase and Lowercase text
LOAD	Load module
OBJ	Object module
PLI	PL/1 program

Fig. 5.4.1.1 Descriptive qualifiers

When a data set is created only the user supplied name has to be specified. The system supplies automatically the identification qualifier and prompts the user for the descriptive qualifier (prompting is the request of the system to the user to supply missing information).
The TSO naming conventions also apply to partitioned data sets. A partitioned data set contains data sets as data. These data sets are called members.
Each member is identified by a member name and can be referred to separately. The member name is enclosed within parentheses and appended to the end of the name of the partitioned data set.

Example 5.4.1.2:

```
        tsotest.alpha.pli(program1)
  partitioned data set name      member name
```

Example 5.4.1.3 shows a list of valid data set names in the manner in which the user might have specified them.

Example 5.4.1.3:

```
tsotest.alpha(program1)
        alpha(program2)
        beta
        alpha.pli
tsotest.gamma
```

5.4.2. Creating and editing a data set

One way to create a data set is by using the EDIT command.

Example 5.4.2.1 :

```
READY
edit alpha.pli(prog1)   new   pli   (CR)
INPUT
00010 example :  proc options (main);   (CR)
00020    (CR)
  .
  .
  .
00150 end example;      (CR)
00160 (CR)
EDIT
save (CR)
SAVED
end (CR)
READY
```

NOTE: For the input of PL/1 programs the user must always leave a blank as the first character of each line.
On an output listing at a terminal the system will insert another blank between the line number and the first character. Thus, in the case of a PL/1 program which is listed, there will appear to be two blank characters at the beginning of each line.

In the above example the user wishes to create a member (calling it PROG1) of a data set (calling it ALPHA.PLI). To do so the user enters the EDIT command with the NEW operand. The NEW operand specifies that a new data set is to be created. The systems responds to this with the INPUT message followed by the first line number. After the user has typed in his first input line the system prints the next line number. In the next line the user wants to insert a space line so he has to type at least one space.
This goes on up to line 00160 where the user strikes the RETURN key immediatly following the line number. This means that the user has reached the end of the data he wanted to enter. The system responds with the EDIT message. The user now has the possibility to use the EDIT subcommands to update, list, delete or to save the data set. After giving the SAVE subcommand the system still remains in EDIT mode. To switch back to the command mode the user types in the END subcommand of EDIT. The system sends the message READY.
The next example shows how to use the CHANGE, LIST, DELETE and INPUT subcommands of EDIT to edit an existing data set called EX.PLI.

Example 5.4.2.2 :

```
edit ex.pli    old   pli (CR)
EDIT
list (CR)
00010   EXAMPLE :  PROC OPTIONS (MAIN);
00020
00030       DECLARE SUM BINARY FIXED(31,0),
00040               I   BINARY FIXED(31,0);
00050
00060       SUM=0; I=1;
00070       DO WHILE (I LE 10);
00080            SUM=SUM+I; I=I+1;
00090       END;
00100       PUT EDIT ('SUM= ',SUM) (A,F(10));
00110
00120   END EXAMPLE;
END OF DATA
change      70 /10/1000/    (CR)
delete      50 (CR)
input       102 2 (CR)
INPUT
00102         put edit ('processing finished') (A); (CR)
00104 (CR)
```

```
EDIT
list (CR)
00010   EXAMPLE :  PROC OPTIONS (MAIN);
00020
00030       DECLARE SUM BINARY FIXED(31,0),
00040               I   BINARY FIXED(31,0);
00060      SUM=0; I=1;
00070      DO WHILE (I LE 1000);
00080           SUM=SUM+I; I=I+1;
00090      END;
00100      PUT EDIT ('SUM= ',SUM) (A,F(10));
00102      PUT EDIT ('PROCESSING FINISHED') (A);
00110
00120   END EXAMPLE;
END OF DATA
save (CR)
SAVED
end (CR)
READY
```

5.4.3 Reserving a data set.

The installation policy of the JRC computing centre is that data sets created with the EDIT command are normally only kept until the end of the session. To obtain a longer reservation the user can use certain command procedures (a command procedure is a set of TSO commands and optionally subcommands which are executed upon calling of the command procedure). One possible way of reserving and creating a data set using a command procedure is shown below.

Example 5.4.3.1:

```
READY
creares ex.pli user01 (CR)
IKJ56234I ATTR-LIST $@#4321 NOT FOUND
IKJ56247I UTILITY DATA SET NOT FREED IS NOT ALLOCATED
EX.PLI
   RECFM-LRECL-BLKSIZE-DSORG-CREATED---EXPIRES--SECURITY
   FB      80      3120     PS  07/07/78  00/00/00    NONE
-- VOLUME --
   USER01
TO TERMINATE, REPLY AT ANY TIME 'END' OR 'STOP'.
DO YOU WANT TO RESERVE, INQUIRY OR STOP?  (REPLY R,I OR S)
r (CR)
SPECIFY AUT.NO.  AND PROGR.NO.
........ (8 NUMERICS)
14550823 (CR)
SPECIFY THE VOLUME SERIAL NUMBER.
......  (6 ALPHANUMERICS)
useroa (CR)
SPECIFY THE DATA-SET NAME (FULLY-QUALIFIED DSNAME).
tsotest.ex.pli (CR)
SPECIFY THE EXPIRATION DATE (DAY/MONTH/YEAR).
```

```
..... (6 NUMERICS)
010875 CR
YOUR DATA-SET IS NOW RESERVED.
DO YOU WANT TO RESERVE, INQUIRY OR STOP?  (REPLY R,I OR S)
s CR
READY
edit ex.pli old CR
EDIT
input CR
INPUT
00010 example:  proc option (main)
   .
   .
   .
```

First the user types in CREARES (the name of the command procedure) giving it as operands the name of the data set he wants to create and to reserve and the name of the volume the data set is supposed to be stored on (the following volumes are available:
USERXX with XX = 0a/0b/0c/0d/0e/0f/0g).
Now the system interacts with the user as shown in the example and finally sends the message READY indicating the completion of the work. The user now types in the EDIT command using the OLD operand and after he typed in the subcommand INPUT things are going the same way as described in section 5.3.2.
To get more information on the usage of the CREARES command procedure the following command can be used:

```
READY
help   creares CR
```

Installation policy with respect to reservation and accounting of data sets is described in the installation notes in the section 'PROC NOTES (LIBRAIRIES PRIVEES),PAG.H.1-1'.

5.4.4 Printing a data set on the line printer

Because it might be very time consuming to list large data sets on the user's terminal, certain command procedures are provided to print a data set on the line printer. The example below serves as a first introduction into the usage of the LST command procedure.

Example 5.4.4.1:

```
READY
lst   ex.pli (CR)
IKJ580I DATA SET $L$S$T.$S$C$R$A NOT IN CATALOG
SAVED
IKJ56247I UTILITY DATA SET NOT FREED, IS NOT ALLOCATED
TO TERMINATE, REPLY AT ANY TIME 'END' OR 'STOP'.
SPECIFY AUT.NO.  AND PROGR.NO.
........ (8 NUMERICS)
14550823 (CR)
SPECIFY BOX NO., JOBNAME SUFFIX AND PROGRAMMER'S NAME
.............(3 NUMERICS, 1 ALPHANUMERIC AND MAX.16
ALPHANUMERICS)
999a--donald duck (CR)
YOUR JOB IS NAMED 'TSOTESTA' AND HAS BEEN PASSED TO HASP.
READY
```

To get more informations on the LST command procedure the following command can be used:

```
READY
help lst (CR)
```

5.5 Programming at the terminal

5.5.1 Allocating data sets

A program in a programming language may be thought of as a realization of a function f which, given as input a set of values called x, delivers as output a set of values called y, hence y=f(x). The output set y is computed by applying the function f to its input set x.
From the point of view of the operating system those sets are called data sets (see section 5.4.1) and are known to the operating system by their data set names.
From the point of view of program f those sets are called files and are known to the program f by their file names.
Now the question arises: How does a program like program f know which data set it has to take as its input file and which data set it has to take as its output file?
Two solutions are possible :

- the allocation is made by name that is the file name must be the same as the data set name,
- the allocation is made through a command, which provides a link between the name of the file and the name of the data set.

The last solution which offers more flexibility is used in TSO and the command that does it is the ALLOCATE command.
But besides establishing the link between files and data sets the ALLOCATE command can be used also to create a data set. This is important with respect to output data sets.

Example 5.5.1.1 :

```
READY
free file(sysin,sysprint) (CR)
READY
allocate file(sysin) dataset(in.data) old (CR)
READY
alloc file(sysprint) da(out.data) new block(800) space(10) (CR)
READY
run ex.pli pli (CR)
.
.
.
READY
free file(sysin,sysprint) (CR)
READY
allocate file(sysin) dataset(*) (CR)
READY
allocate file(sysprint) dataset(*) (CR)
READY
run ex.pli pli(CR)
.
.
.
READY
```

In example 5.5.1.1 a program EX.PLI is executed. Its input file is called SYSIN and its output file is called SYSPRINT.

EX.PLI is executed twice, the first time all input data is taken from data set IN.DATA and its output data is placed in data set OUT.DATA. The second time all input data is taken from the terminal and output is listed on the terminal.

The first FREE command is used to break any predefined file allocation (this might be the case, since SYSIN and SYSPRINT are the standard PL/1 input and output files). The next ALLOCATE command now provides a link between file SYSIN and data set IN.DATA. The second ALLOCATE does the same with file SYSPRINT and data set OUT.DATA. But since the data set OUT.DATA does not yet exist, it must be created. This is done by using the NEW operand. The BLOCK and SPACE operands then tell the system how much storage space is needed, that is 10 blocks with a size of 800 bytes for each block.

After this is done program EX.PLI can be started.

After execution is finished the files SYSIN and SYSPRINT are freed again and now are assigned to the terminal (DATASET(*) is defined to mean the terminal). Program EX.PLI can be started once more.

Section 5.6 shows two complete terminal sessions.

5.5.2 Compiling, link editing and executing a program.

A digital computer can be thought of as a device for scanning and interpreting a sequence of items of information stored in its memory and performing a sequence of actions determined by the information items being scanned. The sequence of items of information being scanned is normally stored in the main computer memory and is referred to as the instruction sequence.

Interpretation of successive instructions of an instruction sequence which constitutes a program is referred to as execution of the program [7].

Programs which can be directly executed by the computer are said to be in internal machine language. However, programs are normally specified by the programmer in some kind of problem-oriented language. To execute these programs they first have to be translated into an instruction sequence in the internal machine language. The translation can be carried out by a computer and is known as compiling. The translation program is called a compiler.

If any program would be translated as a whole and if any translated program was able to be executed immediatly then compilation could be performed as described above. However, in most instances a program consists of several parts which are compiled independently and stored in their translated form for later use. Therefore, compilation does not result in programs written in machine language but in programs written in an intermediate language called object language. The different independently compiled parts of a program are known as object modules. The program which processes object modules to link edit them is called the linkage editor. The output of the linkage editor is called a load module. To place a load module into the main memory for execution a program called loader has to be used. Fig. 5.5.2.1 summarizes the different steps in preparing a program for execution.

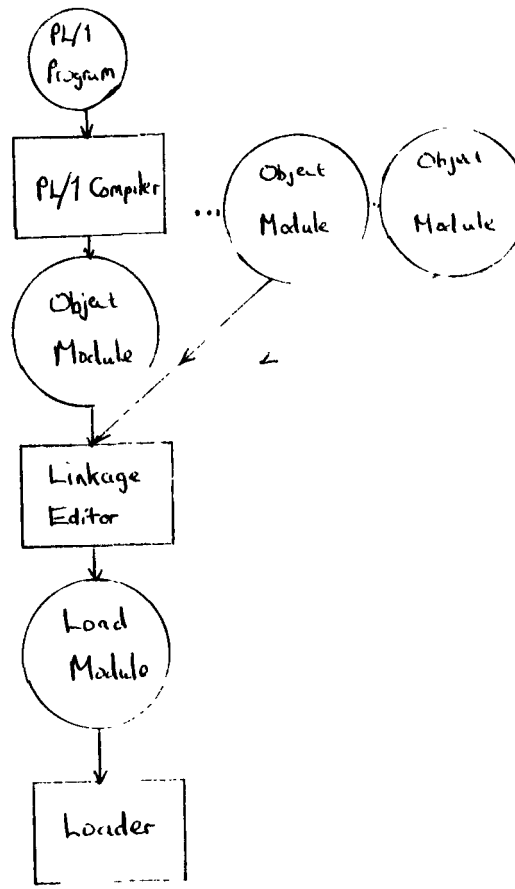


Fig. 5.5.2.1 Preparing a PL/1 program for execution

In TSO several commands are provided to compile, link edit, load, and execute programs at the terminal. TSO also allows the use of other programs, such as utilities, at the terminal. That is, instead of taking a job to the computer room to run it under the operating system, TSO commands can be used to enter it through the terminal. The following examples shows three ways of executing a PL/1 program called EX.PLI.

Example 5.5.2.1 :

```
READY
run ex.pli pli (CR)
      .
      .
      .
READY
```

The RUN command can be used to compile, link, load, and execute the source statements contained in data set EX.PLI (the descriptive qualifier PLI tells the system that the PL/1 compiler has to be used).

Example 5.5.2.2. :

```
READY
pli ex.pli pli (CR)
    .
    .
    .
READY
loadgo   ex.obj (CR)
    .
    .
    .
READY
```

The PLI command calls the PL/1 compiler to compile the program contained in EX.PLI. The compiler produces a data set called EX.OBJ which contains the object module. The LOADGO command then can be used to link, load execute the object module contained in data set EX.OBJ.

Example 5.5.2.3 :

```
READY
pli   ex.pli (CR)
    .
    .
    .
READY
link ex.obj (CR)
    .
    .
    .
```

```
READY
call ex.load(tempname) (CR)
     .
     .
     .
READY
```

The PLI command calls the PL/1 compiler to compile the program contained in EX.PLI. The LINK command takes the object module contained in EX.OBJ (which was produced by the compiler) as input and produces a partitioned data set called EX.LOAD with TEMPNAME as member which contains the load module. The CALL command takes this load module, loads it into main memory and starts execution.

5.5.3 Background processing.

In the TSO terminology a terminal session is sometimes referred to as time-sharing job or foreground job.
Consequently a job that is executed by the batch-processing system is called a background job. TSO now offers the possibility to submit jobs for execution in the batch-processing portion of the IBM System/370 operating system.

Example 5.5.3.1 :

Known: - data set EX.PLI which contains an PL/1 program
- data set IN.DATA which contains the input data

```
READY
pli ex (CR)
     .
     .
     .
READY
link ex (CR)
READY
edit backex.cntl new cntl (CR)
INPUT
00010 //          exec pgm=tempname (CR)
00020 //steplib   dd  dsname=tsotest.ex.load,disp=(shr,keep) (CR)
00030 //          dd  dsname=sys1.plilink,disp=(shr,keep) (CR)
00040 //sysin     dd  dsname=tsotest.in.data,disp=(shr,keep) (CR)
00050 //sysprint  dd  dsname=tsotest.out.data,unit=3330-1, (CR)
00060 //    disp=(new,catlg),vol=ser=useroa,space=(1250,(5)), (CR)
00070 //    dcb=(recfm=vba,lrecl=125,blksize=1250) (CR)
00080 //plidump   dd  sysout=a (CR)
00090 /* (CR)
00100 (CR)
EDIT
save (CR)
SAVED
end (CR)
```

```
READY
free dataset(in.data,ex.load) (CR)
READY
submit backex.cntl (CR)
TO TERMINATE, REPLY AT ANY TIME 'END' OR 'STOP'.
SPECIFY AUT.NO.  AND PROGR.NO.
........  (8 NUMERICS)
14550823 (CR)
SPECIFY BOX NO., JOBNAME SUFFIX AND PROGRAMMER'S NAME.
.............  (3  NUMERICS,  1  ALPHANUMERIC  AND  MAX.  16
ALPHANUMERICS)
999a --micky.mouse (CR)
YOUR JOB IS NAMED 'TSOTESTA' AND HAS BEEN PASSED TO HASP.
READY
   .
   .
   .
READY
IEF404I TSOTESTA ENDED
list out.data (CR)
IKJ52827I OUT.DATA
   .
   .
   .
READY
```

In the above example the user wants to run as background job a PL/1 program contained in data set EX.PLI. As input data set the already existing data set IN.DATA will be used. As output data set a new data set to be named OUT.DATA will be created.
First the user has to produce a load module of the PL/1 program. This he does with the PLI and LINK commands. As result a partitioned data set called EX.LOAD and a member called TEMPNAME is created which contains the resultant load module.
Now the user creates a data set called BACKEX.CNTL (with EDIT) where he puts in the job control statements to define the job. (If the first statement of such a job definition is not a JOB statement, the system generates one when the job is submitted.)
As next step the user frees data set IN.DATA and data set EX.LOAD, to make them available to the background job.
After this has be done the user types the SUBMIT command to submit the job definition contained in data set BACKEX.CNTL to the batch-processing portion of the IBM System/370 Operating System. The user now is prompted to supply additional information needed to complete the JOB card generation. After the following READY message the user is free to do some other work while his job is executed in the background.
Completion of the background execution is announced by the message TSOTESTA ENDED. To obtain the content of the created output data set OUT.DATA the user enters the LIST command.

5.6. Example sessions.

5.6.1 PL/1 session.

The task is to write a PL/1 program which given a positive integer n computes the sum= $\sum_{i}^{n} i$. (To make the user input available to the system every input has be finished by pressing the carriage return key of the used terminal. In the following example it is illustrated by (CR).

```
Turn ON/OFF switch to ON
and then type a(CR)
IKJ54012A ENTER LOGON -
logon tsotest/passall proc(pl1log) (CR)
TSOTEST LOGON IN PROGRESS AT 10:15:50 ON JULY 21, 1978
NO BROADCAST MESSAGES
CPU - 00:00:01 EXECUTION - 00:00:13 SESSION - 00:00:22
READY
edit ex.pli new pli (CR)
INPUT
00010  example:proc options(main);  (CR)
00020  (CR) to enter a blank line type at least one blank
                                (space)      before hitting (CR)
00030  declare sum binary fixed(31,0),  (CR)
00040          i   binary fixed(31,0),  (CR)
00050          k   binary fixed(31,0),  (CR)
00060          n   binary fixed(31,0);  (CR)
00070  (CR)
00080  k=65000;  (CR)
00090  put edit('enter value of n:') (a) skip;(CR)
00100  get list(n); (CR)
00110  (CR)
00120  do while(n gt 0 and n le k); (CR)
00130     sum=0, i=1;        (CR)
00140     do while(i le n);        (CR)
00150        sum+sum+i;          (CR)
00160     end;    (CR)
00170     put edit('n= ',n,' sum= ',sum)(a,f(10),a,f(10));   (CR)
00180     put edit('enter value of n:') (a) skip;     (CR)
00190     get list(n);       (CR)
00200  end;    (CR)
00210  (CR)
00220  put edit('processing stopped on n= ',n) (a,f(10));    (CR)
00230  (CR)
00240  end example;      (CR)
00250  (CR) to leave the input mode and to go back to the edit mode
hit immediatly (CR) without typing any blank (space)
EDIT
save (CR)
SAVED
```

```
list (CR)
00010 EXAMPLE:  PROC OPTIONS(MAIN);
00020
00030 DECLARE SUM BINARY FIXED(31,0),
00040         I   BINARY FIXED(31,0),
00050         K   BINARY FIXED(31,0),
00060         N   BINARY FIXED(31,0);
0007C
00080 K=65000;
00090 PUT EDIT('ENTER VALUE OF N:') (A) SKIP;
00100 GET LIST(N);
00110
00120 DO WHILE(N GT 0 AND N LE K);
00130    SUM=0, I=1;
00140    DO WHILE(I LE N);
00150       SUM+SUM+I;
00160    END;
00170    PUT EDIT('N= ',N,' SUM= ',SUM)(A,F(10),A,F(10));
00180    PUT EDIT('ENTER VALUE OF N:') (A) SKIP;
00190    GET LIST(N);
00200 END;
00210
00220 PUT EDIT('PROCESSING STOPPED ON N= ',N) (A,F(10));
00230
00240 END EXAMPLE;
END OF DATA
end (CR)
READY
pli ex (CR)
PL/I OPTIMIZER V1 R3.0 PTF 65 TIME:  10.28.18 DATE:  21 JULY 78
COMPILER DIAGNOSTIC MESSAGES OF SEVERITY W AND ABOVE
ERROR ID L # NUMBER MESSAGE DESCRIPTION
SEVERE AND EFROR DIAGNOSTIC MESSAGES
IEL0399I E     130       SEMICOLON ASSUMED AFTER 'SUM=0'.
IEL0327I S     130.2     INVALID SYNTAX.        ', I=1' IGNORED.
IEL0304I S 150 INVALID SYNTAX AFTER 'SUM'.  'SUM+SUM+I' IGUORED.
MESSAGES SUPPRESSED BY THE FLAG OPTION:  2 I.
END OF COMPILER DIAGNOSTIC MESSAGES
COMPILE TIME   0.00 MINS      SPILL FILE:  0 RECORDS, SIZE 4051
COMPILATION ENDED BY 'NOCOMPILE' OPTION
READY
```

```
edit ex.pli old pli (CR)
EDIT
list 120 2000 (CR)
00120 DO WHILE(N GT 0 AND N LE K);
00130    SUM=0, I=1;
00140    DO WHILE(I LE N);
00150       SUM+SUM+I;
00160    END;
00170    PUT EDIT('N= ',N,' SUM= ',SUM)(A,F(10),A,F(10));
00180    PUT EDIT('ENTER VALUE OF N:') (A) SKIP;
00190    GET LIST(N);
00200 END;
00210
00220 PUT EDIT('PROCESSING STOPPED ON N= ',N) (A,F(10));
00230
00240 END EXAMPLE;
END OF DATA
change 130 /,/;/ (CR)
list 130 (CR)
00130            sum=0; i=1;
change 150 /+/=/ (CR)
list 150 (CR)
00150             SUM=SUM+I;
input 151 1 (CR)
INPUT
00151            i=i+1; (CR)
00152 (CR)
EDIT
list (CR)
00010 EXAMPLE:  PROC OPTIONS(MAIN);
00020
00030 DECLARE SUM BINARY FIXED(31,0),
00040         I   BINARY FIXED(31,0),
00050         K   BINARY FIXED(31,0),
00060         N   BINARY FIXED(31,0);
00070
00080 K=65000;
00090 PUT EDIT('ENTER VALUE OF N:') (A) SKIP;
00100 GET LIST(N);
00110
00120 DO WHILE(N GT 0 AND N LE K);
00130    SUM=0; I=1;
00140    DO WHILE(I LE N);
00150       SUM=SUM+I;
00151       I=I+1;
00160    END;
00170    PUT EDIT('N= ',N,' SUM= ',SUM) (A,F(10),A,F(10));
00180    PUT EDIT('ENTER VALUE OF N:') (A) SKIP;
00190    GET LIST(N);
00200 END;
00210
00220 PUT EDIT('PROCESSING STOPPED ON N= ',N) (A,F(10));
00230
00240 END EXAMPLE;
END OF DATA
save (CR)
```

```
SAVED
end (CR)
READY
run ex.pli pli (CR)
PL/I OPTIMIZER V1 R3.0 PTF 65 TIME:  11.01.07 DATE:  21 JULY 78

NO  MESSAGES  OF  SEVERITY  W  AND  ABOVE  PRODUCED  FOR  THIS
COMPILATION
MESSAGES SUPPRESSED BY THE FLAG OPTION:  2 I.
COMPILE TIME   0.01 MINS   SPILL FILE:  0 RECORDS, SIZE 4051

ENTER VALUE OF N:10 (CR)
N=          10 SUM=          55
ENTER VALUE OF N:100 (CR)
N=         100 SUM=        5050
ENTER VALUE OF N:11111 (CR)
N=       11111 SUM=    61732716
ENTER VALUE OF N:0 (CR)
PROCESSING STOPPED ON N=          0
READY
IEC130I PL1DUMP DD STATEMENT MISSING
IBM013I SYSUSER NO SUITABLE PLIDUMP DD CARD
edit in.data new nonum (CR)
INPUT
10 100 (CR)
11111 0 (CR)
(CR)
EDIT
save (CR)
SAVED
end (CR)
READY
free file(sysin,sysprint) (CR)
READY
allocate file(sysin) dataset(in.data) old (CR)
READY
alloc file(sysprint) da(out.data) new block(800) space(10) (CR)
READY
loadgo ex.obj (CR)
READY
list out.data (CR)
IEC130I pl1dump dd statement missing
IBM013I SYSUSER NO SUITABLE PLIDUMP DD CARD
OUT.DATA
ENTER VA LUE OF N:N=          10 SUM=          55
ENTER VA LUE OF N:N=         100 SUM=        5050
ENTER VA LUE OF N:N=       11111 SUM=    61732716
ENTER VA LUE OF N:PROCESSING STOPPED ON N=          0
READY
logoff (CR)
TSOTEST LOGGED OFF TSO AT 14:30 :31 ON JULY 21, 1978+--
(turn ON/OFF switch to OFF )
```

5.6.2 FORTRAN session.

The task is to write a FORTRAN program which given a positive integer n computes the sum = $\sum_{i=1}^{n} i$

(To make the user input available to the system every input has to be finished by pressing the carriage return key of the used terminal. In the following example it is illustrated by (CR).

```
Turn ON/OFF switch to ON
and the type a (CR)
Y.M73Z2ZG :  RU:S VI,IR
IKJ53020A ENTER LOGON
logon tsotest/passall proc(fg1log) (CR)
TSOTEST LOGON IN PROGRESS AT 09:29:37 ON AUGUST 1, 1978
NO BROADCAST MESSAGES
CPU - 00:00:01 EXECUTION - 00:00:24 SESSION - 00:00:48
READY
edit ex.fort new fortgi (CR)
INPUT
00010         integer*4 sum (CR)
00020         integer*4  i (CR)
00030         integer*4  k (CR)
00040         integer*4  n (CR)
00050         k=65000 (CR)
00060      10 write(6,70) (CR)
00070         read(5,80) n (CR)
00080         if(n.le.0) go to 60 (CR)
00090         if(n.gt.k)zgo to 60 (CR)
00100         sum=0 (CR)
00110         do 50 i=1,n (CR)
00120      50 sum+sum+i (CR)
00130         go to 10 (CR)
00140      60 write(6,100) n (CR)
00150         stop (CR)
00160      70 format (' enter value of n:') CR
00170      80 format(i5) (CR)
00180      90 format(' x= ',i10,' sum= ',i10) (CR)
00190     100 format(' processing stopped on n= ',i10) (CR)
00200 (CR)      to leave the input mode and to go back to the edit
mode            hit immediatly (CR) without typing any blank (space)
EDIT
save (CR)
SAVED


list (CR)
00010         INTEGER*4 SUM
00020         INTEGER*4  I
00030         INTEGER*4  K
00040         INTEGER*4  N
00050         K=65000
00060     10 WRITE(6,70)
00070         READ(5,80) N
```

```
00080         IF(N.LE.0) GO TO 60
00090         IF(N.GT.K) GO TO 60
00100         SUM=0
00110         DO 50 I=1,N
00120      50 SUM+SUM+I
00130         GO TO 10
00140      60 WRITE(6,100) N
00150         STOP
00160      70 FORMAT (' ENTER VALUE OF N:')
00170      80 FORMAT(I5)
00180      90 FORMAT(' X= ',I10,' SUM= ',I10)
00190     100 FORMAT(' PROCESSING STOPPED ON N= ',I10)
END OF DATA
end CR
READY
fort ex CR
G1 COMPILER ENTERED
00000120      50 SUM+SUM+I
                 $
01) IGI013I SYNTAX
01) IGI015I * NO END STA.
SOURCE ANALYZED
PROGRAM NAME = MAIN
* 002 DIAGNOSTICS GENERATED, HIGHEST SEVERITY CODE IS 8
READY


edit ex.fort old fortgi CR
EDIT
list 120 2000 CR
00120      50 sum+sum+i
00130         GO TO 10
00140      60 WRITE(6,100) N
00150         STOP
00160      70 FORMAT (' ENTER VALUE OF N:')
00170      80 FORMAT(I5)
00180      90 FORMAT(' X= ',I10,' SUM= ',I10)
00190     100 FORMAT(' PROCESSING STOPPED ON N= ',I10)
END OF DATA
change 120 /+/=/ CR
list 120 CR
00120      50 SUM=SUM+i
input 121 1 CR
INPUT
00121         write(6,90) n,sum CR
00122 CR
EDIT
input 191 1 CR
INPUT
00191         end CR
00192 CR
EDIT
list CR
00010          INTEGER*4 SUM
00020          INTEGER*4 I
00030          INTEGER*4 K
```

```
00040          INTEGER*4 N
00050          K=65000
00060       10 WRITE(6,70)
00070          READ(5,80) N
00080          IF(N.LE.0) GO TO 60
00090          IF(N.GT.K) GO TO 60
00100          SUM=0
00110          DO 50 I=1,N
00120       50 SUM=SUM+I
00121          WRITE(6,90) N,SUM
00130          GO TO 10
00140       60 WRITE(6,100) N
00150          STOP
00160       70 FORMAT (' ENTER VALUE OF N:')
00170       80 FORMAT(I5)
00180       90 FORMAT(' X= ',I10,' SUM= ',I10)
00190      100 FORMAT(' PROCESSING STOPPED ON N= ',I10)
00191          END
END OF DATA
save (CR)
SAVED
end (CR)
READY
run ex.fort fort (CR)
G1 COMPILER ENTERED
SOURCEzANALYZED
PROGRAMzNAME = MAIN
*    NO DIAGNOSTICS GENERATED
ENTER VALUE OF N:
00010 (CR)
X=          10 SUM=           55
ENTER VALUEzOF N:
00100 (CR)
X=         100 SUM=         5050
ENTER VALUE OF N:
11111 (CR)
X=       11111 SUM=     61732716
ENTER VALUE OF N:
00000 (CR)
PROCESSING STOPPED ON N=           0
READY
edit in.data new nonum (CR)
INPUT
00010 (CR)
00100 (CR)
11111 (CR)
00000 (CR)
(CR)
EDIT
save (CR)
SAVED
end (CR)
READY
free file(ft05f001,ft06f001) (CR)
READY
allocate file(ft05f001) dataset(in.data) old (CR)
```

```
READY
alloc file(ft06f001) da(out.data) new block(800) space(10)CR
READY
loadgo ex.obj CR
READY
list out.data CR
OUT.DATA
ENTER VA LUE OF N:
X=            10 SUM=           55
ENTER VA LUE OF N:
X=           100 SUM=         5050
ENTER VA LUE OF N:
X=         11111 SUM=     61732716
ENTER VA LUE OF N:
PROCESSING STOPPED ON N=           0
READY
logoff CR
TSOTEST LOGGED OFF TSO AT 10:49:54 ON AUGUST 1, 1978+
turn ON/OFF switch to OFF
```

6. References.

[1] Brinch Hansen, P. : Operating System Principles, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973.

[2] Habermann, A.N.: Introduction to Operating System Design, SRA, 1976.

[3] Corbato, F.J., Merwin Daggett, M., and Daley, R.C.: An experimental time-sharing system, Proc. AFIPS Fall Joint Computer Conf., pp. 335-44, May 1962.

[4] IBM System/360 Operating System: Time Sharing Option Terminal User's Guide GC28-6763-X.

[5] IBM System/360 Operating System: Time Sharing Option Command Language Reference, GC28-6732-X.

[6] JRC - Ispra Computing Centre Newsletter, pp. 16-25, Sept. 1974.

[7] Wegner, P. : Programming Languages, Information Structures and Machine Organization, McGraw-Hill, 1968.