# An approach to executive system maintenance in disk-based systems

*By* R. F. Rosin*

**The use of disk files as the resident store for executive system components has presented particular problems in the areas of system integrity and system maintenance. The former is easily met with conventional techniques, but the latter is more difficult because of the need to optimize disk-allocation at the time the system is edited in order to minimize access time to system components. However, the time required to produce an optimally allocated system edit is prohibitive for debugging purposes. An approach to solving this problem is to develop a quick patch editing facility which can be used at any point in the job stream for debugging purposes, and to use the conventional, slow facility only when a permanent change is needed. A particular implementation of the quick editor is described in detail, and the extension of this kind of facility for time-sharing systems is discussed.**

## 1. Introduction

As disk files become readily available at attractive prices, they are being used more extensively for storage of executive system components. This leads to several potential problems which are outlined in the next section of this paper. The succeeding section consists of a description of a solution to one central problem as it was effected on an IBM 7090/7040 Direct Coupled System. The last two sections of the paper contain a discussion of extensions of this approach and its applications to more general multi-programming systems, and a presentation of results and conclusions.

Disk files have shown several advantages over tape files for storage of system records. The long access time often required to obtain a record in a tape system can, in general, be reduced from seconds to milliseconds. This leads to a second positive feature of disk-based systems in that the number of components in an executive system can grow quite large at little or no increase in system overhead, since access time will remain relatively constant. Of importance, also, is the fact that whereas tapes tend to wear and become unreliable, disk files are quite impervious to such effects. Therefore, the inconvenience and expense of preparing fresh system tapes on a regular basis, often daily in heavily used systems, is avoided.

On the other hand, disk systems do offer certain problems which must be handled differently than in tape systems. These generally fall into two categories, protection and maintenance.

## 2. Potential problems in disk-based systems

### 2.1 System integrity

Protection of a disk system requires that no user be able to modify the system, purposely or inadvertently, thus preserving the integrity of the software. Also, a user must not be able to gain access to, or modify any other user's program or data. Protection in tape systems is accomplished: (1) by making the tape units holding the system records inaccessible to the user, (2) by making the input and output streams one-way (e.g. the input file cannot be backspaced), and (3) by placing a mark in the input stream which only the system can cross. In order to accomplish this, rather elaborate schemes have been devised both in hardware and software to prevent the user from accomplishing certain input-output manipulations. For example, in some hardware, unauthorized attempts at I/O manipulation will interrupt the computer.

In disk-based systems, comparable protection devices must be employed. Since many different kinds of records (e.g. system input, user scratch area, translators, etc.) can exist in the same physical disk file, integrity protection requires that certain tracks, and not tape units, must be removed from the realm of user access and control. This is usually accomplished by partitioning schemes and central I/O software systems similar to those used in tape-based systems. The designer must be careful to preserve flexibility while guaranteeing protection.

### 2.2 System maintenance

System maintenance, the second area of potential problems in disk systems, cannot be treated so simply as it is in tape systems. Compilation of the records constituting the system, establishing the linkage between them, and transforming them into a machine-readable form are mutually similar in both types of system.

However, the format of the system records, the random access characteristics of their eventual storage medium, and the potential size of the system itself can lead to problems in preparation of the system files, a process usually called *system editing*. The first of these problems is that the system records should appear on disk in locations such that the total seek time for any job or set of jobs to be run will be kept to a minimum. If, for example, the monitor and loader are at opposite extremes of the disk, in terms of read-head motion, then it is likely that time will be expended in waiting for certain areas of disk to be physically accessible to a read-head. The editor should alleviate such difficulties.

There exist disk files which allow reading and writing in the so-called "cylinder mode"[1] in which corresponding tracks on adjacent disks are referenced in sequence automatically. This feature allows loading of system components in a chain, with only one seek and one read

* *Department of Engineering and Applied Science, Yale University, New Haven, Connecticut, U.S.A.*

command. This results in great savings over the use of files in which every system record must be chained together from independent tracks, and the chain processed by a software routine. On the other hand, this creates some additional problems in efficient allocation of the system and other records to disk tracks, and, therefore, increases editing time.

To further insure integrity of the system, the editor should produce a back-up tape of the newly generated system. Good practice often dictates that this tape must be *purposely* loaded by an operator to replace the previous version of the system on disk. This allows those responsible for system maintenance to examine the printed results of the edit in order to obtain some degree of confidence in the newly created system before it is used. In many cases it is this same tape which is used as input for the next edit (i.e. this is the tape against which changes are edited).

As the system becomes large, the editing process is quite time consuming. A 7094/7044 Direct Coupled System with only a few components can require many minutes of computer time for such a process. To this time must be added the appreciable time required to load the tape onto the disk after having ascertained the success of the edit.

All of this time can be absorbed moderately well by an installation which uses relatively static software and does not intend to expand its facilities, i.e. an installation which seldom needs to edit. On the other hand, installations which are constantly extending the scope of their software, for example expansion of the subroutine library, find that the time spent in disk editing can be intolerable. This is particularly true during the debugging of a system component, which often must take place in the final system environment and not as a "job in the system". In this latter case a separate edit must be executed each time a set of modifications is to be tested. Proper system organization can alleviate some, but not all, of this expensive processing by allowing complete checkout of a component in other than the final environment.

## 3. One solution to the system maintenance problem

### 3.1 Environment in which the solution was effected

#### 3.1.1 Hardware

The solution to be discussed was developed for use on the IBM 7094/7040 Direct Coupled System (DCS).* In this configuration, the 7094 acts as a slave to the 7040 to the extent that it generally is incapable of any independent input-output. The 7040 monitors the job flow, processing the input stream, output stream and all tape, disk and unit record I/O requests, and the extensive system I/O queues maintained on the disk file. To accomplish this, the transmit (TMT) command of the 7040 has been modified to allow 94/40 transmission on a core-to-core basis at 8 $\mu$sec per word. (The comparable rate using a 7044 is 5 $\mu$sec per word.)

However, there is one feature of this system which is an important exception to the I/O dependence of the 7094. That is the ability of the 7040 to initiate a scatter-read from disk directly to the 7094 without further 7040 intervention. In this way, system records (and, in the cases where appropriate software has been developed, user records) can be loaded into the 7094 in rapid order. This facility is constrained to the extent that the record to be read must be contained in one cylinder and have the necessary channel commands interspersed at the appropriate junctures.

The 7094 and the 7040 have the ability to interrupt each other at will, and, using the modified TMT command, can communicate by means of blocks of code. This is, in fact, the way all 7094 input-output requests generally are processed. To facilitate this communication, since the 7094 does not process a TMT command, upon interrupt the 7040 examines and interprets the contents of locations $22_8$ and $23_8$ in the 7094. Such an interrupt of the 7040 by the 7094 is called a "HEY" and is accomplished by execution of a BTTE (beginning of tape test on channel E) command.

#### 3.1.2. Manufacturer supplied software

DCS software supplied by IBM includes 7094 IBSYS with a modified input-output executor (IOEX) which issues appropriate HEYS instead of actual I/O commands. The IBJOB system, consisting in large part of the monitor, IOCS, FORTRAN-IV, MAP, COBOL and the subroutine library, is supported under IBSYS. Also included in DCS software is DCMUP, the control program which is always present and active in the 7040. It is the latter program which monitors total system activity.

The system editing facilities supplied are rather extensive. The primary facility is a three-phase processor which:

1. Modifies the distributed tape
2. Produces a new system tape
3. Reformats the output for optimum use of the disk.

As mentioned earlier, this can often be time consuming. However, two quick edit facilities are also provided. One allows DCMUP to be loaded with patches which will remain in use until DCMUP is reloaded, thus allowing checkout of new and temporary installation facilities without performing an edit. It is also possible to patch any component of IBJOB on a one-shot basis, again without editing. These facilities constitute a major capability to alleviate the need for time consuming edits as described in Section 2.2.

#### 3.1.3 Additional software

It is assumed here that the vast majority of computing centers are not content with total reliance on software provided by any manufacturer. While one or more of these companies may set the de facto *standards* of the

* Actually, any combination of 7090–94/7040–44 can be considered. No attempt is to be made to describe fully the design of this system. See Ref. 2 for greater detail.

industry, it is not true that they monopolize *leadership* of the computing field. Even the most understaffed computer center adds to the initial subroutine library, while other centers go so far as to ignore completely the programming facilities provided with the hardware.

The situation at Yale, for example, dictated that FORTRAN II had to be made available due to the large number of users with heavy investment in programs written in that language. This is hardly a unique situation in the ranks of DCS installations. Thus, another subsystem at the level of IBJOB was made available. There have been many different approaches to implementing this facility which is illustrated by the fact that Yale had actually two completely independent FORTRAN II subsystems, one of which has been phased out but was necessary early in the change-over from a previously available 709 system.

Most university computing centers and some industrial installations find their users have need of languages which offer more power, flexibility and novelty of application areas than FORTRAN. Yale has added to its total system the MAMOS subsystem developed at the University of Maryland. This system, as modified at Yale[3] includes much of the University of Michigan Executive System facilities[4] (MAD, UMAP, MADTRAN, Library, IPL-V, SNOBOL) and the ALGOL and SLIP languages.

Thus, the Yale Computer Center had an IBSYS system with three subsystems other than IBJOB.* None of these additional systems can be modified or tested using the IBJOB patch facility, yet they are subject to frequent modification. Due to the size of the total system, a full edit, including reloading the newly created system tape, requires about 30 minutes. Clearly, the IBSYS editing and patching facilities are not as flexible as one would like considering the environment.

### 3.2 The solution to Yale's maintenance problem (DEDIT)

#### 3.2.1 Objectives

In the light of the situation just described, it was decided to explore the possibility of providing additional editing facilities to enable modification and replacement of any system record in an expeditious manner. (The question of making a permanent change in a similar way will be treated in Section 4.) Note that the IBSYS facilities described earlier do not allow temporary *replacement* of a component, only *modification* of existing code.

Any editing program written had to run in the normal job flow on the 7094. As discussed earlier, this facility had to ensure the complete safety of the current system and all other jobs in any active queue. For example, a job waiting in the print queue must be processed successfully regardless of any changes being made in the system by a later job. Likewise, all subsequent jobs must be completely unaffected.

* Since this paper was written the IBM Sort, the IBM 360 Support Package and PUFFT (Purdue University Fast Fortran Translator) have been added as subsystems, leaving a total of six.

The ability to conform to IBSYS editing facilities was mandatory, so that changes debugged in the temporary patch editor, called DEDIT, could be incorporated into a full system edit without modification.

IBJOB, MAMOS and one of the FORTRAN II subsystems all operate as described in Sections 2.1 and 2.2. The second, since discarded, FORTRAN II system did not follow the I/O conventions of DCS, i.e. employ HEY codes. As a result, it was accommodated into the system as a non-standard subsystem and was not stored on the disk in a form to be loaded directly into the 7094. It was decided that the new editing facility should handle only "direct-loading" subsystems since the majority of installations use only this form of subsystem and the only non-direct-load subsystem at Yale was soon to be abandoned.

#### 3.2.2 Implementation strategies

In order to discuss the strategy which was used, a few details of DCMUP (the 7040 control program) must be explained. In order to locate a system record on the disk, DCMUP has available in core a double entry table consisting of pairs of names and initial track addresses of system components in the order they would appear if on a system tape. Also included are all-zero word pairs to indicate file marks, and "*EOT" to indicate the end of the table. Non-direct-loading components have negative track addresses, and, therefore, can be distinguished easily from those the editor is to be able to process. A backup copy of this table is available on track 0 of the disk.

What must be done in order to change the system is: (*a*) prepare a new cylinder (or portion of a cylinder) with the desired component, and (*b*) change the track address in the corresponding entry in the system name table in DCMUP. Using the IBSYS job concept[5] which allows processing several segments serially in one job with integrity of I/O files preserved between segments, one can first edit and then execute programs to evaluate the changes made. To guarantee system integrity three constraints must be applied. First, DCMUP itself cannot be changed (except for the system name table). Second, any change to the system name table must be deleted at the end of 7094 job processing by reloading the original table from disk. Third, any information written on disk by DEDIT must not endanger any files which are not completely processed, such as in the input and output queues.

The first constraint is handled easily by code in DEDIT. The second requires that DCMUP sense the end of a job in which DEDIT has been used and reload the system name table from disk, presenting a minor problem since the editor is to run on the 7094.

The final constraint is overcome by allocating 10 disk cylinders (400 tracks) permanently as user scratch areas and for DEDIT, making them unavailable for system records and queues. (These tracks are available to the user through appropriate scatter read-write routines in the subroutine library of each subsystem.) Since 10

cylinders were reserved, the editor has the power of modifying up to 10 system components in any job.

As mentioned earlier, direct-loading system records contain channel commands indicating the initial 7094 address and block length of each element. Each command appears immediately preceding the element it is to load. A final termination command immediately follows the final element. It was decided that preparation of very long elements would be much more cumbersome than construction of the new system record on a 460 word track-by-track basis. Therefore, every track being prepared by DEDIT has an initial channel command with a word count of 459 or less.

Records being modified are read in and written out immediately on a track-by-track basis until the terminal channel command is encountered. At this point any modifications are chained onto the record in sequence as described in the preceding paragraph. Records thus modified can be loaded for testing in future segments with modified locations being overwritten as loading proceeds into the newly extended chain of channel commands and elements. This strategy allowed development of DEDIT in a straightforward manner without the necessity of retaining large blocks of information in the 7094 during editing.

Finally, it was decided that the card formats and deck structures used in DEDIT should be compatible with those used in the standard edit. Therefore, DEDIT control cards are of the form:

| | Col. 7 | 16 |
|---|---|---|
| | *REPLACE | NAME |
| and | *MODIFY | NAME |

where NAME is the name of the system record to be processed. The octal patch cards are also identical to those used in standard DCS editing with the exception of an optional feature described below.

### 3.2.3 Programming of DEDIT

The implementation of the new editing facility was carried out in two phases; the development of a new HEY code and its interpretation in DCMUP, and the editor itself which runs on the 7094.

The new HEY code has several functions.* Through appropriate calling sequences it can be used in 7094 programs to:

1. Obtain the 7040 address of the first location of the system name table in the 7040. Use of this command also sets a switch which is tested in DCMUP at the end of every job and, when set, initiates reloading the system name table from disk;
2. Transmit in either direction between the 7094 and 7040;
3. Read or write a selected track in the reserved area of the disk.

The 7094 program is illustrated in the accompanying flow chart (see **Fig. 1**). Note in block I of the figure that octal patch cards (indicated by "OCT" in columns 8–10 of a BCD card) are preprocessed and then treated as if the information had been on a binary card. If the loading address specified on such a card is 0 (zero) or blank, and if it does not immediately follow a *REPLACE or *MODIFY card, it will be chained into the previous information.†

Block II includes routines to detect the presence of a full track of information, which results in filling in a necessary word count in the previous channel command and setting up a new channel command with the appropriate loading address.

The facilities indicated in Block III of the figure allow the user to obtain intermediate output including input card images, track images and indications of the stages of processing as they occur. This has been a most valuable aid during debugging and does not affect execution of the program if not used. On the other hand, it is expected that this facility will enable other DCS installations to adopt DEDIT for their own use in relatively short time in spite of different conventions for track allocation, etc.

If the result of Block IV is *MODIFY or *REPLACE, then the address of an unused cylinder is obtained for preparation of a new system record.

The organization of this program also allows additional control cards to be added easily in the future, if desired.

Debugging took place in three phases. DEDIT was first written in the MAD language, and debugged using temporary routines to simulate, on the 7094, subroutines which would later use the new 7040 HEY code. When the HEY code was implemented and the MAD program was debugged, the assembly language program capable of using the HEY was prepared and debugged by deleting the four simulation routines, one at a time, from the MAD program. Finally, using the MAD program as a flow chart, a MAP program was written to implement DEDIT in IBJOB for distribution. (FORTRAN IV implementation would have been impractical due to the necessity for the logical manipulation of word fields.) It is felt that this technique of debugging in a high level language reduced the checkout time by a large factor.

### 3.2.4 An example of use of the editor

The following sequence effects the replacement of IPL-V in MAMOS, and also a modification of the MAMOS monitor. This is followed by two segments to test these changes.

```
$JOB
$ID                          identification information
$EXECUTE   IBJOB
$IBJOB
    DEDIT deck
$DATA
```

* The author wishes to acknowledge the work of Mr. James Blanchard who carried out the 7040 HEY code programming.

† This optional feature is the only departure from standard DCS editing input deck structure.

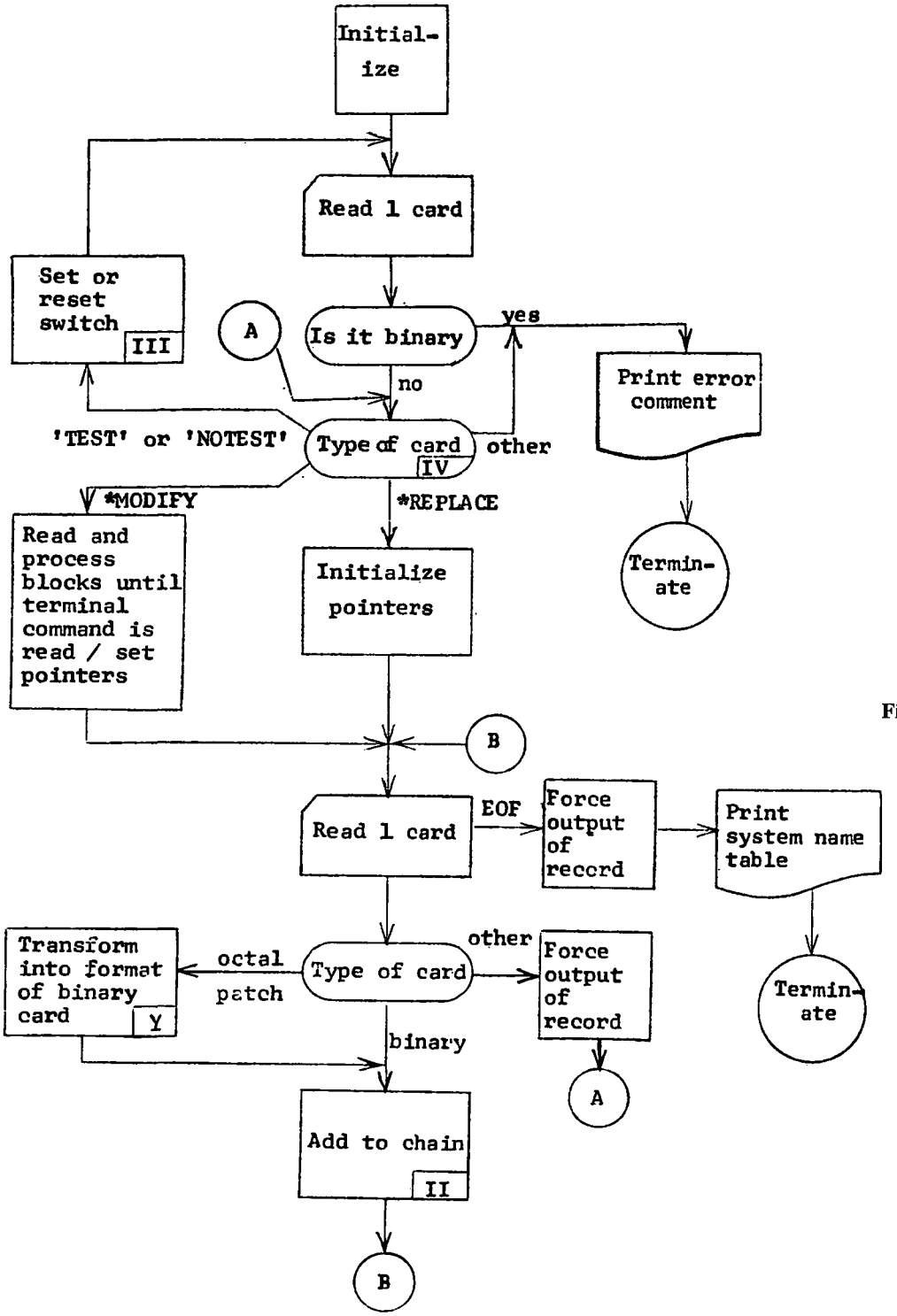


Fig. 1—Flowchart of DEDIT

*REPLACE IPL-V
Binary and/or octal deck
*MODIFY MADMLI
Binary and/or octal patches
$EOF
$EXECUTE MAMOS
Test segment 1
$EOF
$EXECUTE MAMOS
Test segment 2
$EOF
$JOB (next job in input queue)

Unless octal patch cards with the optional feature of zero

or blank location fields are used, the replacement and modification control cards and the binary and/or octal patch cards which follow may be used unchanged as input to a standard DCS edit.

### 4. Extensions

The addition of *TAPE to the control card repertoire of DEDIT could be used to initiate the creation of a self-loading system tape representing the system as modified. However, as useful as this would probably be, there would be the accompanying disadvantage of adding the time-consuming process of optimum track allocation to the editing process. At present, it is envisioned that DEDIT will be used for debugging of new system components, and the distributed editor will be used for system tape preparation; i.e. for instituting a new version of the operational system.

The ability to delete and insert system records is a possible extension to the new editor, but not a necessary one. Replacing a given record with a dummy segment in order to simulate deletion is trivial in the context of DEDIT. Insertion can be simulated by replacing a record not needed in the checkout effort with the desired new record and by making a corresponding change in the control section which will load this record. The necessity for these features would arise only if *TAPE were implemented.

Although the 7094/40 DCS has some multi-programming and multi-processing features, multi-console time sharing creates new problems with respect to the approach presented here. It is likely that two or more users will require access to the same (re-entrant) translator on an interleaved schedule. In this case, during use of this new editor, each user will need his own copy of the system name table, again to ensure integrity. Restoration to normal processing, i.e. a universally used system name table, would be included in the purging of any job using this new type of editing facility. It is likely that the programming effort required to implement such a scheme would not be so great as to overshadow the flexibility which this scheme affords. The ability to develop and debug system software in the normal job flow is a very positive addition to any system.

### 5. Results and conclusions

The approach described here was implemented in under two months on the 7094/7040 DCS at the Computer Center, Yale University. It requires less than 2 seconds per track written to complete an edit before attempting to test the results. This time contrasts with the minimum of 30 minutes typically required to edit a new system, test it and then reload the current system tape. In typical cases, a factor of 50 is realized in terms of time saved.

As in the case of compilers, it appears that a set of editors of differing capabilities and characteristics is desirable in a software system. When optimum disk usage is required as in the fully operational system, then the time spent to achieve this efficiency through optimum track allocation is generally well spent. However, for short debugging and analysis runs, quick processing more than makes up for comparatively inefficient use of disk storage since the situation is in effect for a very short period of time.

### References


1. IBM Corporation. "General Information Manual, 1301 Disk Storage with 7631 File Control", Form D22-6576.
2. Smith, E. C. "A Directly Coupled Multiprocessing System", *IBM Systems Journal*, September–December 1963.
3. Rosin, R. F. "MAMOS Under IBSYS at Yale", Computer Center, Yale University, 1965.
4. *University of Michigan Executive System for the* 7090 *Computer*, University of Michigan Press, Ann Arbor, 1964.
5. IBM Corporation. "IBM 7090/7094 IBSYS Operating System, System Monitor (IBSYS)", Form C28-6248.


## Correspondence

*To the Editor,*
*The Computer Journal.*

Sir,

I should like to comment on the points raised by Dr. Samet in his letter "Progress?" in the August issue of the *Journal*. We can indeed be proud that so many of the techniques of programming were originated on EDSAC 1 and Pegasus machines, but the question is to what extent this lead has been maintained. The significant thing about the use of relocatable binary in the EGDON system is not that a new technique has been invented, but that it is being used in a British computing system in a way that has been common in American systems for many years. The reason for using relocatable binary is not that it provides relative addressing, but that it makes it possible to divide a large program into subprograms (possibly written in different languages) that can be compiled independently, with a consequent saving of compiling time when a change is made to one sub-program. There is no point in doing this on a small computer with small programs, but on a large computer with large programs the gains are substantial. It is a sad commentary on the development of computing in British Universities that until recently we have had no large computers, and consequently no experience of really large programs and the techniques required to deal with them. Now that we have at last got some large machines we must take care that we do not allow small-machine techniques and ways of thinking to persist.

Yours faithfully,


D. W. Barron

The University Mathematical Laboratory,
Corn Exchange Street,
Cambridge.


5 August 1966