Program conversion—One successful paradigm

by CHARLES LYNN, JR., JEAN RISLEY and ROBERT WELLS

Bolt Beranek and Newman, Inc.
Cambridge, Massachusetts

The continuing rapid development of computer hardware over the years has been both a blessing to the user who sees his computing power increase and cost decrease over time, and a disappointment to those who have found that the cost of converting existing software outweighs the advantages of the new machines. For any proposed change of hardware, there are at least three options whose cost the user must carefully weigh—(1) the option of remaining with older hardware, (2) the option of completely redesigning/rewriting existent software and (3) the option of converting those portions of the existing software which are portable and rewriting the rest.

At BBN we have recently carried out the transfer of a large body of programs from one hardware environment to another. In the process of this task, we have developed an analytical approach to the problem of conversion of programs written in a partially independent higher-level language. Our technique consists of several distinct phases:

1. Identification of the programs to be converted and the environment in which they will be run.
2. Isolation and analysis of those language features which are not portable and of those features of the hardware and system environment which were used.
3. Design and implementation of a mechanical conversion process.
4. Creation and use of a procedure for giving human attention to special or difficult areas.
5. Conversion and debugging of the programs including definition and use of standards for debugging and program testing.
6. Creation of the operational environment under which the programs will be run.
7. Parallel operation of both systems with extensive comparison of their respective results.
8. Operational use of the converted system and the archiving of material from the old system.

Dividing the problem in this way makes it possible to keep control of the progress of the conversion effort. Also, performing a complete analysis before actually beginning to convert and run programs provides a better grasp of the scope of the work and tends to minimize surprises later in the project.

THE PROBLEM

Our specific problem was to move all of the current Management Information Services (MIS) functions of the company from the GE Timesharing Service* to an in-house DECSYSTEM-20. Our primary motive was the reduction of the continually growing hardware cost in terms of machine time and storage charges. Those costs were much higher than those projected for the in-house machine, and the prohibitive cost of increasing service and development of new software effectively precluded major improvements. We were also constrained by the absolute requirement for continuity of performance—the change of environment had to be transparent to the company.

Throughout the conversion project, there were a few basic policies. The primary constraint was that the converted system behave “exactly” like the old one. This provided a strict guideline on program rewrites—nothing was to be rewritten if it could be done in the old way.

The body of programs comprising the MIS system to be converted consisted of approximately 450 source program files (70,000 source lines of code) written in three different dialects of FORTRAN (*FORTY*, FIV and F77). There were both batch and interactive processes, and 15 hierarchical data bases with a total of 65 record types were used by the programs. The system was developed almost entirely by internal BBN MIS development personnel over several years, with some support software such as the data base management system provided by the time-sharing vendor. Program documentation was generally limited or out of date, with most documentation consisting primarily of user instruction memos.

ANALYSIS

During the analysis phase of the conversion, we divided the potential language and environmental conversion problems into specific analysis areas, with responsibility for one or more areas assumed by each member of the team. The main areas were (1) character and string handling data types.*
and features present in each of the GE FORTRAN dialects but absent from the DEC FORTRAN: (2) data base issues (HISAM hierarchical database on GE, relational databases provided by System 1022 on DEC); (3) input/output and file handling language features; (4) system-provided features (subroutine packages, monitor commands, SORT/MERGEs, interrupt error handling, etc.); (5) other language feature incompatibilities (ENCODE/DECODE, initialization statements, structured programming features, etc.); and (6) other facilities used, such as the database report and batch stream command languages.

All the source code files were initially transported to the DEC system where they were kept on-line continuously. During the analysis phase we used the vendor-supplied manuals to locate potential incompatibilities in the languages, and then searched the existing on-line sources for actual occurrences of any potentially troublesome syntax.

We specifically designed a search program to accept a set of search patterns and a set of exclusion patterns, specified as regular expressions, and scan a specified group of files for matches, which were then listed with their locations. Exclusion patterns were used to eliminate non-matching files, such as those found within comments. This search program and all the programs used to mechanically translate the FORTRAN sources were written in FASBOL, a compiled dialect of SNOBOL for the DEC System-10 and DEC-System-20 systems. SNOBOL is a powerful string processing language, and it proved ideal for the rapid development of these conversion programs.

The information collected during the analysis phase was circulated within the whole MIS group by an on-line message system (HERMES). This was useful both for communication of specific points ("NAMELIST I/O is never used," "there is a lot of EQUIVALENCE with character data variables," etc.), and to provide on-line hard copy of problem areas by topic as a guide for later phases. There were group reviews and discussion sessions on the findings in each area based on the collection of messages. This information was also circulated to the development group so that they would be aware of language features that should be avoided in their development work.

MECHANICAL CONVERSION

The design of the mechanical conversion itself began with listing, by area, the specific mappings of syntax which would change a feature in one of the GE FORTRANs to acceptable DEC FORTRAN. For example, in an unformatted read, map `READ,...` to `ACCEPT *,...'; or in a type statement with initialization values, `remove the initialization values and create a DATA statement for the variable.' Specific translations were grouped into 17 processes, each of which became a FASBOL program.

Each conversion process was conceived as a filter, taking an input file and applying a set of translations to it. The intermediate forms between these processes were not, in general, legal FORTRAN programs; for example, one of the initial filters concatenated continuation lines together to make lines hundreds of characters long so that later filters would not have to be concerned with continuation lines. Another filter analyzed each routine to develop symbol table information about the variables used, then distributed this information as control character sequences to all occurrences of the variables. This allowed later filters to trivially determine such information as the type and dimensionality of any variable. The last FASBOL filters in the sequence removed such alterations to produce valid FORTRAN programs. Due to the largely unstructured nature of SNOBOL programs, we felt it would have been much more difficult to develop only one large SNOBOL program to perform all the translations; most of our filter programs were under three pages long and understandable as a whole.

The ordering requirements of the translation processes, shown in the graph of Figure 1, evolved based on special requirements of the processes themselves. For instance, the process to reconcile OPEN, CLOSE and unit assignments (OPCLOS) needed to occur before the one resolving problems of format modes and expressions in I/O lists (IOLIST) in order to locate FORMAT statement numbers in the READS and WRITEs. Figure 1 also shows, for example, that the relative order in which SHORT and STATE were executed does not matter, but both require LINES to have been run and both must be run before any of the other routines (DBCALL, IOTRIV, or ENVIR) may be run. The execution order which was actually used for the 17 modules is indicated by the numbers in the upper left corner of the boxes in Figure 1.

For many features, a choice had to be made between conversion to acceptable DEC forms and emulation of the GE-provided features. In most cases we favored conversion, since we hoped to minimize long-term maintenance problems caused by the holdover of alien conventions. However, the use of the HISAM data base calls and the representation of character data in strings was so difficult to convert to DEC that it was decided to emulate these functions in the converted system.

The HISAM hierarchical data base structure so dominated the logical structure of the programs that conversion of the programs to make optimal use of the System 1022 relational data base facilities would have required almost total restructuring and rewriting. We therefore decided to emulate the HISAM data base routines where possible rather than undertake extensive rewriting. The HISAM emulation consists of 14 external routines and several internal routines, comprising 1700 lines of source code. The package was written in RATFOR, a structured FORTRAN pre-processor, because of the readability and cleanliness of the structural statements it provided.

Character processing in all the FORTRAN dialects, including the target DEC language, is very dependent on the number of characters that fit into single- and double-word variables. The GE code was firmly based on four characters per word and could not be practically mapped into the five-character words that are used for literals and support routine calls on DEC. We therefore provided a minimal preprocessor (MISFOR) for long-term maintenance of the code. This
Program Conversion—One Successful Paradigm

GE Fortran Programs

1. SPLIT: create file for each logical program; reformat line numbers

2. LINES: lower case to upper; comment out JCL; remove spaces; " to "; remove continuation lines

3. SHORT: compress 8 character names to 6 by table or truncation

4. STATE: split one-line IF; IF..THEN..ELSE; DO WHILE/UNTIL; end DO with CONTINUE; split a=b, ...

5. DBCALL: remove data base open & close; translate calls to data base routines

6. IOTRIV: rewrite terminal IO statements (READ, ACCEPT, WRITE, TYPE)

7. ENVIR: rewrite ABENDS, STOP, monitor calls; note SORT

8. NAMES: collect variable type, length, dimension information; tag each variable

9. DATAST: create DATA statements for initialization in type statements; pad strings

10. ENDECO: rewrite ENCODEs & DECODEs; preceed DECODEs with call to DECOER routine

11. OPCLOS: associate file name with unit #; note direct access 10; build new OPEN & CLOSE

12. IDLIST: create variable for constants in 10 list; remove BN, $m., $n.; 4 character per word strings

13. TEXT: character assignments, concatenations, substrings become subroutine calls

14. PREFOR: mark 4 character per word literals; compress IF statements

15. CLEAN: collect & order statements; remove symbol table info; build continuation lines

16. COMBIN: recombine subroutines into single file

17. MISFOR: TEXT4 \rightarrow INTEGER; expand 4 character per word literals with blanks

DEC Fortran Programs

Figure 1—Execution precedence graph of translation processes.
preprocessor trivially translates a data type declaration line (TEXT4 to INTEGER) for variables containing four character/word data and translates syntactically-marked literals by inserting a blank every four characters so that DEC FORTRAN will effectively store four characters per word. Support routines have also been provided to convert four-character-per-word strings to the five-character-per-word strings needed for interfacing with DEC system routines.

Many capabilities that were language features on the GE system became calls to subroutines from the newly-created support library for the converted code. For example, the error return on DECODE, character assignment and comparison statements, and the INQUIRE for file status all became subroutine calls during the mechanical conversion.

MANUAL CONVERSION

Once the mechanical translation had been completely specified and developed, there remained a list of problems that required manual intervention. These included cases of more complicated syntax which were not used often enough to justify complete treatment in the translation programs, as well as issues such as changing file name conventions or sort command syntax which were not simply decidable. The mechanical translation both listed these troublesome items and placed a comment before them in the source program.

For the programmer performing the manual checking and editing there were a number of reference materials and memory aids. First, there was search program output with line number locations of specific potential problems, (SORT, overlays, possible bit fields, PROCEDURE calls, monitor calls, etc.). Second, there was the output of the mechanical phase with error messages marking possible problems. Third, there was a summary document and a complete specification of subroutines in the support library with calling conventions. Finally, there was a step-by-step document listing all issues expected to arise during the manual conversion. These reference materials were the primary force for completeness and consistency during the manual conversion, and they made it possible for each team member to recreate the conversion environment even when there had been distractions to other tasks.

PROGRAM CONVERSION AND DEBUGGING

For conversion, the program set was divided into subgroups of 10 to 30 programs each, usually by similarity of function or access to the same data sets. These subgroups were generally subsets of the functional groups defined by the operational system. The fact that most of the programs are organized around file input and output made it relatively straightforward to isolate each program. The actual conversion of programs tended to consist of four steps—(1) a preliminary (sometimes optional) manual edit, (2) the mechanical conversion, (3) a finalizing manual edit and (4) extensive debugging and testing. In general, each person tended to convert 10 to 20 related programs at once, moving a whole group of programs through all of the steps. There was specialization within each group, with particular project members being most expert at understanding the complexities of sort conversion, data base usage, or data base report language usage. However, each member was responsible for conversion of groups of programs from beginning to end, with consultation if needed with the appropriate specialist.

During the manual conversion we discovered that some repeated sections of code occurred sufficiently often to merit inclusion in the support library. The requests for interactive input (TYPE, FORMAT, ACCEPT, FORMAT) were so frequent that we created functions to receive literal messages and return a value of the appropriate type. Similarly, a function was added to locate and format the date for use in many of the programs. In general, however, we tried to minimize the number of subroutines introduced so that our efforts in the conversion phase could be concentrated on conversion rather than tool development.

While most of the actual program conversion was straightforward, a few problem areas were encountered. Some of the programs required history or parameter files which had not been transferred to the new system; short files were usually re-entered manually while longer ones were transferred via tape. The time-sharing system used for the first part of the conversion (before our own DEC SYSTEM-20 arrived) crashed frequently; usually only a few minutes work was lost although at one point it crashed, was down for a week, and somehow managed to lose several additional days work.

The most troublesome problems involved the parallel modification of programs. One situation involved GE programs which were modified between the time they were transferred to the new machine and the time they were actually converted; checking the dates on which the programs were last modified immediately before beginning to convert a group of programs prevented conversion of outdated modules. A second situation involved GE programs which were modified after they were converted. When this happened, the modified program was either again transported to the new system where it was mechanically compared to the earlier version, or else the development personnel making the GE modifications also modified the DEC version. The latter procedure, when not carefully controlled, led to a third situation—two people modified a converted program at the “same” time. This problem was later avoided by making each program the responsibility of one individual and making sure that no one else modified it without first checking with the program’s nominal owner.

As each program was converted it was debugged. Debugging generally began with synthetic input and then proceeded with copies of actual input from the running system. The use of synthetic data made testing of programs which performed input validation and editing operations much easier since the actual input data was voluminous and would rarely, if ever, contain all possible errors. The use of large amounts of actual input from the running system made validation of the computations performed by the programs easy to check since the correct answers were already available.
It should be noted that when a discrepancy was found, it was not always because the converted program was wrong; we found several bugs in the running system. The action to be taken in such cases required careful consideration. Repairing the operational system was not always desirable—in one case it required that many people change the way values were interpreted. Therefore, the converted program was modified to run in a "simulate-the-bug" mode. In cases where numbers which were printed on reports were truncated (without any indication by the GE system) the field widths were increased. Since the data in the reports was not generally re-entered into the system, these differences did not lead to further discrepancies. In several cases, bugs in the operational system were fixed.

A disturbingly large portion of the debugging time was consumed by bugs, some amounting to a comedy of errors, which were not related to the conversion. Bugs in the DEC SORT utility provide many examples. One of its numerous errors was traced to a record in a sorted file whose record length was changed by the sort. After a couple days we were told that the bug would be fixed in the next software release. We could get it shortly but it required the next monitor release because it used new monitor calls to parse the new command string format. However, that version of the monitor was not scheduled for release for two months. Thus, all the calls to the sort utility had to be found and changed to conform to the new syntax and a routine had to be written to emulate those functions of the new monitor which the new sort used.

**BATCH STREAM DEVELOPMENT**

Most jobs on the GE system were run in overnight batch mode. The control file was built and submitted by a program which used a prototype and operator responses to simulated program queries. In addition to running programs, the streams created temporary backup copies of some files (it was too expensive to backup all of the files and data bases) and performed other file maintenance operations. The command language made built-in recovery mechanisms so hard to implement that none were used. Development personnel were frequently required to perform manual recovery operations when a service interrupt occurred during a job since time did not permit restoration of files from the normal GE dumps (that would require waiting until the next night). The batch streams were totally rewritten because of the large differences in the command languages and operating environments and because of the desire to improve error detection and recovery mechanisms.

The operations environment we developed on the DEC system is similar to the GE system in many ways. Batch control files are built from prototypes and operator responses; specially marked strings in the prototype, the questions, are replaced by the responses. Several advantages have resulted from running all programs from batch streams: the operations staff is free to use their terminals for other tasks, a log is automatically created for each job, and the running environment is consistently defined. Since space is available, all major files and databases which are to be modified by a job are first copied; "deleted" files are only logically deleted. All intermediate steps are saved on the daily dump tapes before being physically deleted from the system. In the event of a crash or other large error it is relatively easy to backup to any specified point.

**PARALLEL OPERATION**

The validation of the system by running in parallel was a formidable task. The operational system worked under tight time constraints for data input processing and report distributions. Originally, the system had a fairly large component of interactive data entry and report request and retrieval. We made a few operational changes preparatory to running in parallel, replacing interactive data entry with batch entry, in order to get better control over input and changes to the system.

The parallel operation phase evolved into two parts. During the first part all inputs to the operational system, usually in the form of card decks, were collected and transferred via magtape to the new system. Every half-month (the company business cycle) the operational data bases and transaction files were dumped to tape and loaded onto the parallel system. About one day was required to read the tapes (11 2400-foot reels) and build the binary transaction files and System 1022 data bases. Batch control files were then generated from the stream prototypes based on the GE operations log and then run (overnight) in the same order as they had been run on the actual system. This operation, in addition to testing the individual programs, tested the batch streams, their interaction with the programs (mostly file management), and our ability to recover and backup to any specified point. While some bugs were found, most required excessive amounts of time to locate since the actual bug could have occurred anywhere within the simulated bimonthly period.

These problems led to a second mode of parallel operation during which the parallel system was run as close to the operational system as possible (typically within a day) with the reports being checked line-by-line (rounding differences of one cent were usually ignored). In order to isolate differences in data files as quickly as possible, two techniques were developed. The first was a detailed RECAP program, used on both systems, which calculated transaction file summaries. After a job stream was run, RECAP was used on the modified files; comparison of the outputs quickly isolated the differing transactions. The second technique was a complete record-by-record match of the contents of all modified data bases and some transaction files performed by a set of batch jobs and programs to sort and run file comparisons. A match was run weekly during full parallel operation.

Many of the differences encountered during the parallel operation were of a mechanical nature. Getting the same input data for both machines was a significant problem. The problem of dates had been foreseen and a modification to the DEC system date routine was made to allow the date to be pre-set. This was not sufficient, however, during the
latter stages of the parallel operation when the parallel system was run ahead of the operational one. Date discrepancies in runs made before or after midnight were common. This later resulted in several comparison mismatches which required further manual investigation to ascertain that the differences were only due to the differing dates.

Card input was also a problem. Before the card reader for the DECSYSTEM-20 arrived (about five months late) card decks were sent to a service bureau for transfer to tape. Cards were occasionally lost or out of order. Later, a program was written to allow direct inter-machine transfer of ASCII files. Its 300-baud transfer rate was satisfactory for small card decks but was too time-consuming for larger ones. When the card reader finally did arrive, the data became worse since dark spots on the backs of cards were interpreted as punched holes by GE’s card reader but not by DEC’s. Correcting the consequences of these differences was time-consuming and frequently led to the necessity of additional correction runs. These, in turn, led to comparison differences caused by differences in the unique batch number assigned to each run. This problem was solved by writing programs to selectively zero-specified fields in the files before the comparisons were made.

From the inception of parallel running the original operational personnel played an increasingly important role in the conversion, advising us on how to make the system easier to use, identifying problems in the parallel runs, and, during the second part, conducting the parallel runs. Their enthusiastic cooperation in the conversion was invaluable both in the technical side of the conversion, and in familiarizing them with the new system in advance of the actual switch to it.

PERSONNEL AND SCHEDULING

Our conversion team consisted primarily of four people full-time for 12 months, with expert consultations by several people including one of the original authors and one of the current operators of the system. None of the conversion team had significant previous MIS or business programming experience, but came from scientific, communications, or systems programming backgrounds.

In selection and support of personnel for a conversion project, our experience shows that a careful separation between conversion and development teams is very important. Some interaction is necessary to avoid conversion of obsolete segments and to understand more obscure sections, but, in general, keeping the conversion team separate from design issues permits complete concentration on reproducing the current functions. This tended to prevent a distraction into the history of the code, an involvement with all of the things the code could or should have been made to do, and any dependence on unverified premises about how portions of the code ought to interact.

We had two major overall design and scheduling checkpoints, one before actual conversion began and one halfway through the conversion effort. We decided initially to spend half of the time (six months) in analysis, approach design, tool building and experimentation using one of the functional groups of the system, and at the end of that period the conversion procedure was essentially complete. The second half was originally scheduled to convert all the programs in three months, and run in parallel for three months. Actually we converted about 2/3 of the programs in three months and did both conversion and running in parallel in the final three months. Converting programs, even with mechanical aids, was made more difficult by having to switch between conversion and other support tasks such as installing system software on the new machine or transporting test data and new source files to the new machine.

RESULTS

When the switch to operational use of the new system occurred, all of the regularly-scheduled programs and batch streams were operational. A few quarterly reports and reports which run irregularly on a demand basis had not been converted. A request for one of these reports caused it to be placed at the top of the queue of programs to be converted.

The conversion was essentially transparent to users of the new reports; except for the addition of report numbers and/or page numbers the actual reports are identical. The conversion was not as transparent to those who distribute the reports for mechanical reasons. Since the old system printed several reports interspersed with the batch log they were partially identified by their position in the printout. On the new system each report is printed separately so the report numbers must be used. This was initially a problem since reports which differed by either paging, sort order, class inclusion or exclusion, or run date had been given the same report number; addition of version numbers to these reports solved the problem.

The conversion was least transparent to those people, such as the contract and personnel administrators, who had performed interactive data base updating under the old system. During the parallel operation phase all updates were batched and performed by the operations staff. This method of operation has been retained, at least initially, under the new system until the scheduling and security requirements are resolved. In addition, several people had interactively requested reports under the old system; they must now request the operations staff to run these reports.

For the operations staff the day-to-day running of the system is structurally quite similar to the old system—only a few of the names have changed. The biggest difference involves having to run and distribute both those reports which were formerly requested interactively and the batched update results. The latter must be carefully scheduled since they have to be verified, and frequently corrected and resubmitted, by the people who are responsible for the data.

It is difficult to make direct comparisons of execution times or costs between the two systems due to a large number of unknown factors. The old system provided the user with two quantities—time of day, in hours and minutes, and ‘‘cost.’’ The cost is an undisclosed function of at least
CPU, memory and I/O usage. The new system also provides
the user with two quantities—elapsed time and run-time,
both expressed in hundredths of seconds. The direct and
secondary effects of multiple users in a time-sharing envi-
riment cause these quantities to increase with load but
correction factors are not known. In general, the overnight
runs on the old system required less wall clock time than
the corresponding overnight run on the new system but the
difference does not seem to be "significant" in the opera-
tions environment. This relationship can be reversed under
appropriate loading conditions caused by additional users.

A complete cost analysis of the conversion project is
difficult because many costs cannot be directly compared
and several are not readily quantifiable. New development
on the GE system directly increased costs; on the DEC
system it only increases overall machine usage. The 4800-
baud remote station was critical for the GE system and
communication difficulties frequently delayed operations.
After a service interrupt there was no way to quickly catch
up. The DEC system does not require remote communica-
tions. After an interruption of service the operations staff
can preempt the development group until things are back on
schedule. Things can be scheduled more efficiently on the
DEC system as system loading is more predictable.

Ignoring the factors mentioned in the last paragraph,
a rough estimate of the cost of the GE services was about
$26,000 per month. This includes subscription fees, job and
storage charges, communication costs, remote station lease
and maintenance, shipping charges, etc. The costs associ­
ated with the DEC system are about $13,000 per month.
This figure includes two full-time operators, system hard-
ware (appropriately depreciated) and software (including
the SORT and System 1022 data base packages), mainte-
nance, disks, tapes, cabinets, power and space. The one-time
costs of the conversion from GE were about four person-years of
labor and $65,000 per computer usage (additional GE usage
due to the conversion and portions of the initial mechanical
conversion which occurred before the DEC SYSTEM-20 was
delivered). It is estimated that the conversion will pay for
itself within three years.

CONCLUSIONS

We found that only a very few programs actually needed
extensive rewrites. Some rewriting was required, for ex-
ample, when either there were not enough I/O channels to
access files in the same order as before, or because the
emulation routines were prohibitively slow for the reorga-
nization of some data bases.

The creation of standards for those aspects of the pro-
grams which can change is a process which has continued
throughout the conversion. We found that the creation of
interactive entry conventions applied consistently made the
programs easier to read and debug without changing actual
user interaction. Conventions for file and program names,
Once created and consistently applied, made program and
data grouping more reasonable. Conventions and routines
for accessing and processing dates, for example, provided
a basic consistency between programs that made functional
differences easier to understand. In deciding to institute
each of these standardizations, there was a careful weighing
of alternatives in which the additional cost of implementa-
tion had to be shown to be negligible and the benefit to be
considerably positive.

If we were to begin the project again, there are a few
improvements we would make in the procedure. Attention
would be given to the final operations environment before
report naming and identification conventions were estab-
lished. The HISAM emulation would have been written so
that data base extensions could be made without having to
modify the emulation package and recompile all programs
using it, Time should have been spent to develop an effi-
cient, reliable method of inter-machine data transfer. The
tools to compare binary files, data bases and reports allow-
ing error tolerances to be specified should have been devel-
oped sooner. Finally, ample time should have been allowed
for the "unexpected"; about one-quarter of our time was
devoted to finding bugs in purchased software, installation
of new releases, delays in delivery, finding ways around
problems which could not be readily fixed and system

The overall policy for a system which is undergoing con-
version should be to restrain new development as much as
possible until the conversion is finished. Even fixing bugs
in the old system can be a problem for conversion if the
fixes are not made to the converted programs as well. Any
new development brings up a number of difficult issues such
as whether to develop on the old or new machine, when to
convert if developed on the old machine and how to run
operationally on the new machine before conversion is done.
We have been only partially successful in avoiding these
problems, but without a deliberate management policy of
only allowing the most essential changes, the conversion
would have been made more difficult if not impossible.

Based on our experience, the reliable conversion of a
large body of code from one hardware environment to an-
other is possible with a detailed analytical approach. We
hope that as a result of our experience others will find
conversion to be less of an intimidating, uncontrolled proc-
cess and more of a task for which some reasonable, effective
procedures exist. Conversion may be both expensive and
difficult, but it is possible.

REFERENCES

1. FORTRAN SFORMATS Mark I11 GCO8 Background Service Reference
Manual (2200.01A). Information Services Business Division, General
2. FORTRAN IV (FIV/PEN) Mark I11 Foreground Service Reference Man-
ual (3102.13A), Information Services Business Division, General Electric
Company, Rockville, Maryland, December 1976.
3. FORTRAN Reference Manual (AA-4158B-TM), Digital Equipment Cor-
poration, Maynard, Massachusetts, April 1977.
4. FORTRAN/77 (F77) Mark I11 Foreground Service Reference Manual
(3106.01A), Information Services Business Division, General Electric
Company, Rockville, Maryland, March 1977.
5. Griswold, R. E., J. F. Paige and I. P. Polonsky, The SNOBOL 4 Pro-


