1973

Magnetic Tape Portability

James Alf Iverson

Report Number:

73-101

MAGNETIC TAPE PORTABILITY

James Alf Iverson Jr.
Purdue University
CSD TR 101

MAGNETIC TAPE PORTABILITY

A Thesis

Submitted to the Faculty

of

Purdue University

by

James Alf Iverson, Jr.

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 1973

ACKNOWLEDGEMENTS

Most important of all I would like to thank my wife for her patience, understanding and keypunching during the research and writing of this thesis.

I offer particular thanks to Professor Jay Nunamaker for his guidance, to Professor Maurice Halstead for his professional interest and support and to Dr. T. Aird for his assistance in establishing a direct use for this research.

Special thanks also go to the staff of the Purdue University Computing Center for providing an excellent environment for this research as well as outstanding support.

The initial design effort was partially supported by the National Science Foundation and the detailed design was partially supported by the Purdue University Computing Center.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Iverson, James Alf Jr., Ph.D., Purdue University, August, 1973. Magnetic Tape Portability. Major Professor: Jay Nunamaker.

Magnetic tape portability problems still exist after nearly two decades. This thesis identifies and suggests solutions for several of these incompatability problems. The specific problem addressed is that of producing a procedure whereby a magnetic tape generated by a source computer can be transformed into a magnetic tape acceptable to a target computer.

The method chosen for examining this problem is the design of a user oriented tape transformation algorithm. The algorithm is not dependent on any specific source computer and is limited only by physical constraints on the target computer and its associated tape drives. Tables and algorithms provide the mapping capabilities from one data type to another and from one computer to another. The algorithm does not encompass all of the problems discussed in the thesis.

Difficulties arising when reading and writing magnetic tapes are discussed with proposed solutions. A suggestion for data description is presented along with some operational guidelines for creating and transforming magnetic tapes. A seven step procedure is given for transforming a magnetic tape.

The salient features of the algorithm are embodied in a Fortran implementation on the CDC 6500 in which the machine dependent routines are well identified. A collection of twenty test cases, originating from fifteen different groups, are synthesized to illustrate current problems in magnetic tape portability. The source computers were manufactured by IBM, Control Data, Modcomp, Siemens, Univac, Honeywell and DEC. In all cases but one the target computer was the CDC 6500.

CHAPTER 1

THE NEED FOR DATA BASE PORTABILITY

GENERAL COMMENTS

The computer user of today has access to more data than ever before. He would like to be able to extract information from a data base without being concerned with the complexities of how the data is stored. The computing center, as a result, is becoming more involved in handling large amounts of data. They too would like to keep the user from having to understand how the data is stored. This allows the computing center to make changes to the system that involve the addition or removal of devices such as drums, discs or tapes with a minimum of upheaval. In essence then the goal of both the user and the computing center is to make the mass storage devices transparent to the user.

The current use of higher-level languages such as Cobol, Fortran, PL/1, Algol, Jovial, etc. has made the task of program conversion to different computers much easier. Programs moved from one computer configuration to another have fared rather well providing that they were not programmed to be machine-dependent. On the other hand, the transferring of large amounts of data has not been as successful since methods to describe data have not met with universal approval.

THE AGE OF THE USER

The typical user of computers today has less total knowledge of the computer system than ever before. This is due to the increasing complexity of computer systems as well as the method in which the user is being educated in the use of computers. Over a decade ago, the user was familiar with one higher level-programming language as well as understanding the principles of assembly language and machine architecture. The typical user today is conversant primarily in higher-level languages. He has been taught to use the computer as a tool and the mass storage devices are transparent to him. This lack of knowledge hurts the user when he is faced with the problem of obtaining media, such as cards or magnetic tapes, that contain useful data but do not conform to his current computer system.

In particular, a user requesting data from other computer installations most often cannot control either the format of the data or the storage media on which it is placed. If he purchases the data, a predescribed format often machine and system dependent already exists. The typical user assumes that if he is sent the data in a computer readable form then it can be read on all computers. It is a prevalent notion that magnetic tapes can be read without difficulty on any computer the same as punched cards can be read by any card reader.

This device transparency is beginning to backfire on computer centers. Since the average user believes that all media is easily transferable, he has no regard for the difficulties involved in moving large amounts of data between non-compatible computers. Education as well as assistance is needed for the user.

DEFINITIONS AND TERMINOLOGY

The following definitions are used:

field

the smallest unit of data to be processed. It may be as small as one bit or as large as one or more characters.

<u>logical record</u>

a series of fields related to a common subject.

<u>file</u>

a collection of all of the records of a similar type.

<u>data base</u>

a collection of one or more files that reside in a form acceptable as input to a large scale digital computer.

<u>physical record</u>

this is the data that resides between two physical end of record gaps on a magnetic tape. On some systems this is referred to as a block. It may be composed of one or more logical records.

<u>portable</u>

to be easily carried.

<u>portable data base</u>

a data base that may be easily carried to another computer system and used at another site. It is in a form acceptable as input to a large scale digital computer.

transformation

The term transformation has been chosen to represent the conversion or translation from one data form to another. The terms conversion and translation are not used since they have different meanings for IBM System 360/370 users.

source computer

the computer on which the original data base was created.

target computer

the computer on which the data base containing the transformed data is to be used.

THE INCREASING USE OF DATA BASE MANAGEMENT SYSTEMS

A large number of data bases today are created and accessed using Generalized Data Base Management Systems (GDBMS). GDBMS have evolved throughout the years from the early report writers and report generators. Although developments in GDBMS have taken place since 1958 (Minker 1969), considerable effort has been expended in in the last four to five years.

State of the art reviews have been provided by Climenson (1966), Minker and Sable (1967), Shoffner (1968), and Senko (1969). The proliferation of GDBMS is best illustrated in the reports by Mitre (Fry, et al. 1969), Diebold (1969), CODASYL (1969) and CODASYL (1971). These reports discuss various GDBMS in different depths. Among the GDBMS discussed are DM-1 (Auerbach), Forge (Burroughs), Cogent III (Computer Sciences), IDS (General Electric, now Honeywell), Mark IV (Informatics), GIS (IBM), NIPS/FFS (National Military Command System), TDMS (SDC), Manage (SDS, now Xerox), Rapid (U.S. Army), Disk Forte (Burroughs), IMS-DL/1 (IBM), UL/1 (RCA now UNIVAC), GIM (TRW), ISL-1 (Information Systems Leasing), MIS (National Brewing), Omnibus (Industrial Indemnity), FMS (Pacific Gas and Electric), DF-2 (International Minerals), ADAM (Mitre), and SC-1 (Western Electric). In each of these reports, there is no attempt to define a GDBMS but rather to examine common or distinct attributes of the various systems.

The major efforts by the CODASYL Systems Committee in producing both "A Survey of Generalized Data Base Management Systems" (CODASYL 1969) and "Feature Analysis of Generalized Data Base Management Systems" (CODASYL 1971) represent the works of many individuals with company support and again illustrate the importance that many people and companies are currently placing on being able to access data. Also operating under CODASYL, initial efforts in proposing a language for the description and manipulation of a data base have been presented by the Data Base Task Group (DBTG) (DTBG 1969 and DBTG 1971). DBTG (1969) made some attempt towards standardization but leans heavily towards COBOL and attempts to design its "works" within COBOL. DBTG (1971) has extracted some of the COBOL oriented requirements and placed them in separate chapters. The problem of portability is solved by allowing the Data Manipulation Language (DML) to default to the host language and operating system which in turn handles the physical manipulations of data in primary and secondary storage.

In parallel with the DBTG effort, Guide and Share (IBM user groups) worked together to form a set of future requirements for Data Base Management Systems (Guide-Share 1970). Work at Purdue University on data portability is continuing as part of a project involving the interface of data bases and application programs (Nunamaker et al. 1973).

In retrospect then, the user is demanding a way to better manage his data. Software developers are slowly responding to the needs of the user.

MEDIA FOR DATA PORTABILITY

There are several types of media that can be used for data portability. Some of the more common are: paper(input/ output forms), punched cards, paper tape, data communications, disk packs and magnetic tape..

a) paper - Data is often recorded on data sheets of various sizes. The data may have been printed by some device or may be in hand printed form. This data can then be transcribed directly to cards, magnetic tape or discs. If the data and the data sheet are in a machine readable form, the data can be directly read by optical character recognition devices or mark sensing devices. The codes for these graphic symbols are automatically generated by the particular device and the user does not generally have to worry about them.

b) punched cards - Card decks ranging from a handful to several trays are in common use today. However, they are not generally used for large data bases because of their bulk. Several types of cards are presently in use. The 80-column Hollerith card provides the most universal method for data portability today. A representation of selected graphics characters by specific combinations of punched holes per column has developed into a de-facto standard. As a result, both ANSI Fortran and ANSI Basic Fortran include a representation of their character sets in 80-column punched cards that are widely used in the USA. Some card readers have the capability of reading only either the 96-column cards in use on the lower range of IBM computers or the 90-column cards which have been used by UNIVAC. Exceptions to standard graphic codes arise when the data is in a compacted or compressed form. Compressed cards in their simplest form can be in binary images either by row or by column. They may also be in a more highly compressed form using encoding techniques. The card reader may also influence the format of compressed cards.

c) paper tape - As a general rule, in the United States, this media is not widely used to transport other than small amounts of data. The major users of paper tape are those with special purpose minicomputers equipped with only paper tape input-output. Numerical control machines also are sometimes driven by paper tape.

d) data communications - With current technology it is possible to transmit large amounts of data via communication lines. Generally some well defined interface, either hardware or software, handles the conversion to the proper graphic code for the target computer. The recipient of the data is well aware of what he is to recieve.

e) <u>disk packs</u> - Within a computing center disk packs may be compatible across several computers. They are not generally compatible between different computer manufacturers. Physical constraints such as sectors per disk require different types of drives (i.e. the incompatibilities between the CDC 854 and the IBM 1311.)

f) <u>magnetic tape</u> - Magnetic tape is rather universally used. In particular, 7-track 1/2 inch tape recorded in densities of 200, 556 and 800 bits per lineal inch and 9-track 1/2 inch tape recorded in densities of 800 and 1600 bits per lineal inch are in very common use. It provides a reasonable recording density at a low storage cost.

The major disadvantages of the previously discussed media are as follows. Cards are bulky for large amounts of data. Input forms, unless they are in machine readable form, involve another step introducing human error to place them in computer useable form. Paper tape is not generally available at all installations, and it is fairly bulky. The use of data communications implies that the recipient of the data be aware of its format particularly if it is being used in a real time basis. Disk packs are not always physically and logically compatible. Magnetic tapes are sensitive to dust, temperature and humidity.

For amounts of data consuming more than a few thousand cards, magnetic tapes are still the most physically compatible media. Solving the problem of magnetic tape portability is a first step in solving the general problem of data base portability.

MAGNETIC TAPE PORTABILITY

One of the major problems in data portability today is the task of being able to read magnetic tapes on a computer system that is different from the one that generated them. Barring physical constraints such as density, number of tracks and the width of the tape, the user has had a great deal of difficulty in reading tapes that were not generated on a very similar system to his own. This problem has existed since the early days of computing. However, then most users knew enough about computer systems to solve the problem themselves. The majority of computer installations have solved part of the magnetic tape portability problem with utility packages that allow the user to dump the contents of a tape so that he may examine it more closely.

The remaining part of the problem consists of being able to properly describe the data and its method of being recorded. To date, neither a list of requirements nor a specific algorithm exists for properly handling this problem. In the past, special purpose programs were written, used to transform the data, and then relegated to a corner of a desk. Documentation almost never existed for this "magic deck" and it was redeveloped again and again by each user who had to transform his data. At the present time this procedure still has not changed. A partial solution to this problem is to create a tape transformation program that is general enough to allow a user to transform tapes irrespective of their contents as long as he can properly describe them.

From the user's point of view, the tape may contain programs and data that have been developed and gathered at a considerable cost. He wishes to take advantage of this previous effort and convert the programs and the data to his computer. The cost of conversion will normally be far less than that of developing the program or gathering the data from scratch again. Unfortunately a large class of users are not very sophisticated in their knowledge of computers. In particular, it has been found that users do not generally know the format of the tape that contains the data that they wish to use. Thus we have a frustrated user who has a tape full of data and does not have the expertise to read it or properly describe its contents. This presents a serious problem in that the less information that is known about the tape, the more difficult it is to transform.

In general, there are three classes of users who will send data or programs on magnetic tape.

1) Those who know what they are doing

2) Those who do not know what they are doing

3) Those who partially know what they are doing.

In the first class, the tape sent is either directly useable on the target computer or it is directly transformable to a tape that will be useable on the target computer. This is true barring any physical or mechanical constraints. The user is aware of the incompatibilities of magnetic tapes and attempts to produce a useable tape for the target computer.

The users in the second class are non-programmers. They believe that magnetic tapes are fully compatible irrespective of the computer or operating system. They use or ask someone else to use the capabilities of the source system to produce a tape. They are not capable of describing the format of the data on the tape.

The third class of users produce what can be referred to as "almost" tapes. They understand how to use their source computer reasonably well and assume that everyone else does too. Unfortunately, if the recipient has a computer different than the source computer he may be in for a great deal of trouble.

RECENT INVESTIGATIONS

Throughout the years, there have been concerted efforts to resolve the general problem of portability. The Fortran language was a first step in the direction of providing portable programs. The developers of the Cobol (Common business oriented language) language in using the term "common" felt that they were providing some degree of portability. However, the standards that are set for higher-level languages and the standards observed for higher-level languages are not always the same standards. It is very difficult for users with different requirements to agree on standards that will be beneficial to all regarding portability.

There have been several groups and individuals interested in solving the portability problem. A brief overview of their efforts is presented.

The "Program Transferability Study," (Mealy, et al. 1968) was devoted to the software aspects of transferability. The study considered that problems in program transferability were caused by two major factors. They concluded that current technology encourages or forces the programmer to make implicit in the form of his program:

1. The details of its initial operating system, computer characteristics, etc.

2. The structure and representation of the data accessed by the program.

The study suggested three approaches for attacking the problem:

- administrative control
- extensions to the existing system base
- advanced transferability environment

Administrative control provides checks for adequate documentation and restricts non-standard options. This is a function of the amount of control one can exert over programmers, systems analysts and systems designers. It varies from installation to installation and directly depends on the individuals responsible for enforcing the controls. Extending the system base implies modifying existing systems to allow data descriptions, minimum hardware configurations, text editing, and mechanized administrative control. This is a function of the extendability of the existing system and capabilities of the system programmers. Both approaches have existed in varying forms for many years. The advanced transferability environment proposes to develop prototype systems that will provide transferability. This has not been accomplished to a satisfying degree and it is here that the effort must be placed in order to advance the state of the art.

Gosden (1968) reviewed the status of software compatability as of that time. He examined what had been promised, what existed and what was needed. He too pointed out that one of our current goals should be inter-software compatability. In his short discussions of data exchange and data pooling he expressed the need for a standard data description language. This language would be compatible with data descriptions in current programming languages and provide for easier data interchange.

At the Spring Joint Computer Conference in 1969, a panel session of Software Transferability was held. Five short papers were presented by Ward, Bemer, Gosden, Hopper and Sable (Ward, et al. 1969). The two most significant comments were that (1) data exchange is becoming increasingly important and that (2) programs must be designed for transfer a priori.

The report, "On Program Transferability," (Sattley, Millstein and Warshall, 1970) also discusses program transferability. The report is general in nature and they do not provide solutions to the problem. Of interest is their feeling that the physical characteristics of secondary memory hardware are too diverse to have common features, and that the process of going from an abstract model of data structures to the allocation of hardware resources must be machine-dependent. They recommend modularity as an aid to program transferability but feel as Mealy et al. (1968) that when the logical structure of the program must be modified to accord with stringent mapping instructions, then the possibility of easy transfer disappears.

The most recent group to show some interest in portability is the ACM Special Interest Group on File Description and Translation (SIGFIDET). Its membership includes representatives from industry, government and education. The SIGFIDET (1970, 1971, 1972) workshops have generated discussion and papers that address the problems of file translation and file description.

Groups at the University of Pennsylvania and the University of Michigan have also been investigating the area of data description and translation (French, et al.

1971, Smith 1971, Fry, et al. 1972, Sibley 1972). These efforts will be examined and discussed in Chapter 2.

Some portability does exist throughout a specific series of computers, such as the IBM System 360/370 series, Honeywell 6000 series, Univac 1100 series and the Control Data 6000 series. The level of portability may vary widely however. When changing to a similar computer configuration within the same series, timing problems occur, undiscovered bugs arise and machine dependent features are found. These may result in minor inconveniences or major problems during conversion. Portability problems inevitably become severe when a major change in system or manufacturer is necessary.

The main problem addressed in this thesis is the transfer of large data bases between computers. Chapter 2 will deal with various models for data portability. Problems that arise in magnetic tape portability along with a synthesis of actual cases are discussed in Chapter 3. In Chapter 4, an approach for solving the problem of portable magnetic tapes will be discussed. An actual implementation on the CDC 6500 is discussed in Chapter 5. Chapter 6 presents operational guidelines for magnetic tape interchange. Conclusions and further research areas are discussed in Chapter 7.

CHAPTER 2

MODELS FOR DATA PORTABILITY

GENERAL COMMENTS

Several organizations have been investigating parts of the data portability problem and designing corresponding models. The approaches to the models are presented and briefly discussed. The general problem of data portability is then addressed with three methods, one of which is recommended, for possible implementation.

SOME APPROACHES

The Data Base Task Group (DBTG 1969, DBTG 1971), University of Pennsylvania (French, et al. 1971, Smith 1971) and University of Michigan (Fry, et al. 1972, Sibley 1972) have all addressed the notion of data portability.

Data Base Task Group

The Data Base Task Group proposes both a Data Description Language (DDL) and a Data Manipulation Language (DML). The DDL is used to describe an entire data base or portions thereof. A schema is introduced to completely describe an entire data base by use of DDL entries. A sub-schema, also consisting of DDL entries, is introduced to describe the portions of a data base known to one or more specific programs. The relationship between a schema, sub-schema and a data base management system is illustrated in Figure 2.1.

The DML is the language used to transfer data between a program and the data base. The DML relies on a host language such as Cobol to provide a proper framework for interfacing with the data base. Therefore the inadequacies of the host language for manipulating data in primary storage are perpetuated. The DDL is character oriented since Cobol was used as the original host language. For example, it is not possible to properly describe the negative representation for floating point numbers in the DDL. Such limitations preclude the use of the DDL and DML to transform non-character oriented data.

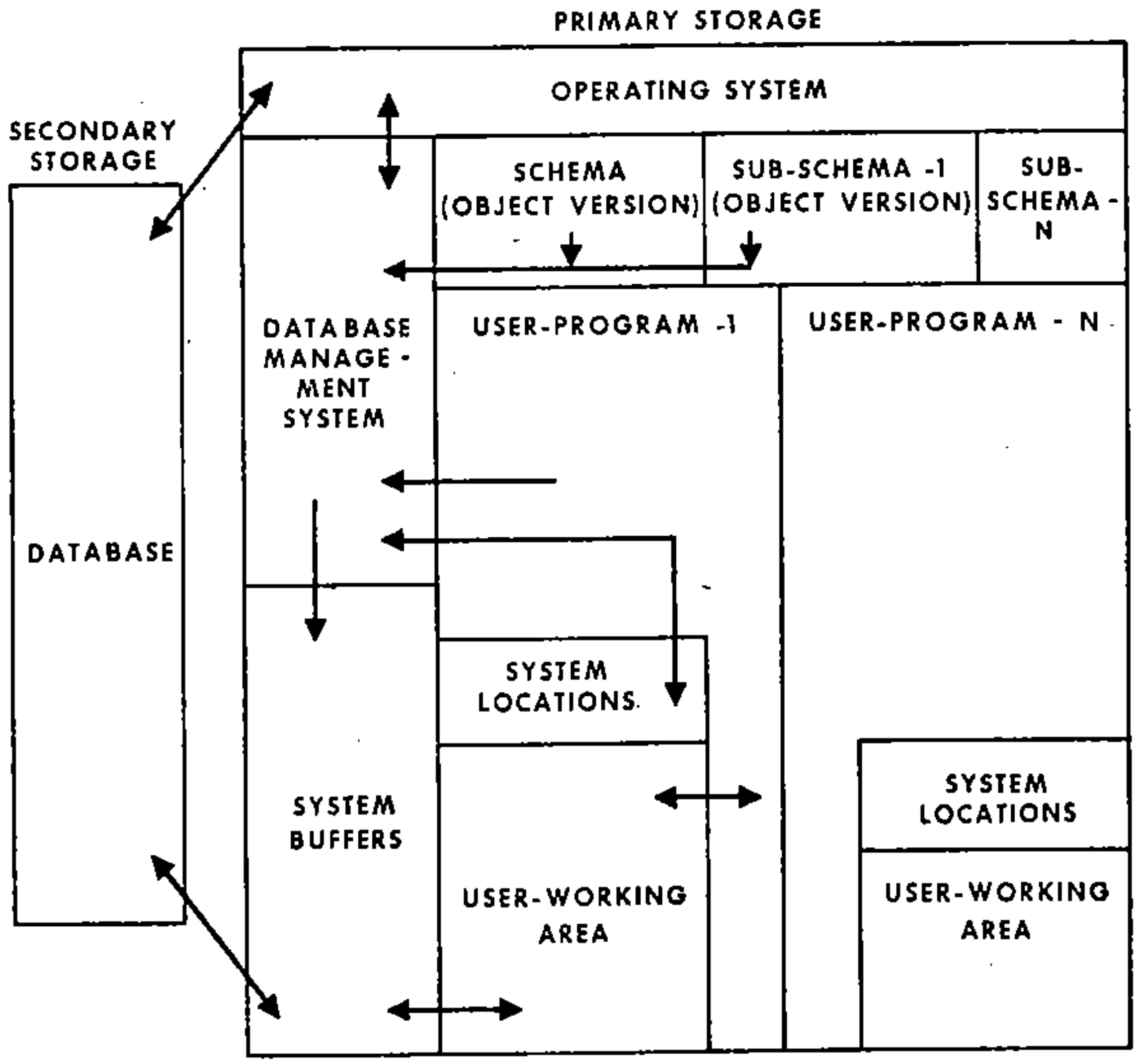


Figure 2.1 DBTG Conceptual Data Base Management System [1]

[1] Reproduced from [DBTG 1971, p. 16]

<u>University of Pennsylvania</u>

Smith (1971) attacks the problem of data description and conversion by defining a Generalized Data Description Language (GDDL) based on a model that characterizes current data organization techniques and provides a framework within which new data structures can be defined. The GDDL is in greater depth than that conceptually proposed by the DBTG. The Smith model for extracting data items from source files and creating target files from target data items is illustrated in Figure 2.2.

The GDDL as described is not general enough to allow a method to properly define the characteristic of a floating point number. In addition, the character set specified for GDDL contains characters that are not available on all computers. Since an implementation is not discussed in this report, it is likely that the GDDL will be somewhat machine dependent upon implementation thus restricting the general model.

A Data Description Language Processor System is also under development at the University of Pennsylvania (French, et al. 1971). The design of the system has been directed at satisfying two requirements of data interchange, (1) data (organization) definition and (2) data translation. The processor system consists of a compiler generator that uses

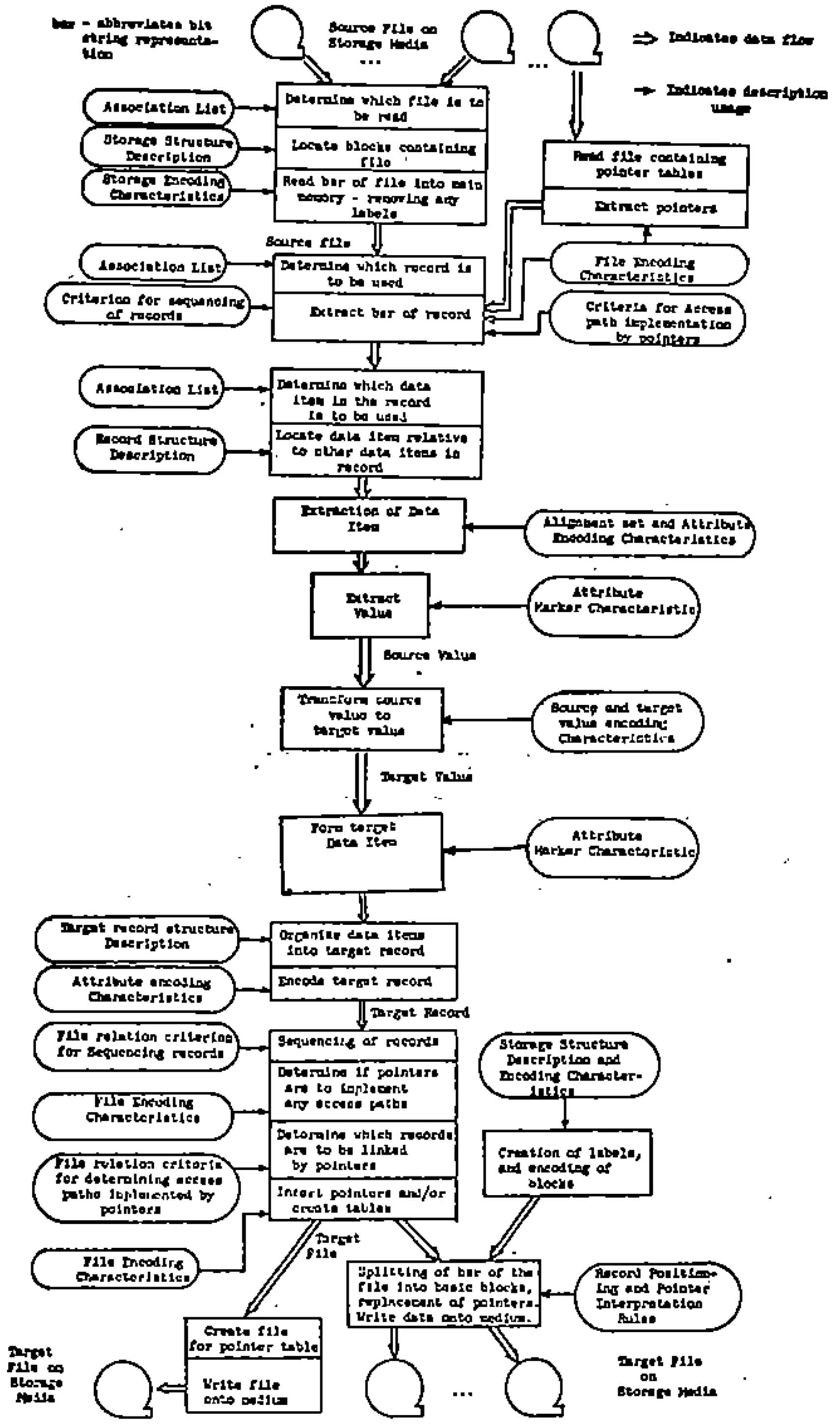

bsr - abbreviates bit string representation
Source File on Storage Media
Indicates data flow
Indicates description usage
Association List
Storage Structure Description
Storage Encoding Characteristics
Determine which file is to be read
Locate blocks containing file
Read bsr of file into main memory - removing any labels
Read file containing pointer tables
Extract pointers
Source file
Association List
Criterion for sequencing of records
Determine which record is to be used
Extract bsr of record
File Encoding Characteristics
Criteria for Access path implementation by pointers
Association List
Record Structure Description
Determine which data item in the record is to be used
Locate data item relative to other data items in record
Extraction of Data Item
Alignment set and Attribute Encoding Characteristics
Extract Value
Attribute Marker Characteristic
Source Value
Transform source value to target value
Source and target value encoding Characteristics
Target Value
Form target Data Item
Attribute Marker Characteristic
Target record structure Description
Attribute encoding Characteristics
Organize data items into target record
Encode target record
Target Record
File relation criterion for Sequencing records
File Encoding Characteristics
File relation criteria for determining access paths implemented by pointers
File Encoding Characteristics
Sequencing of records
Determine if pointers are to implement any access paths
Determine which records are to be linked by pointers
Insert pointers and/or create tables
Storage Structure Description and Encoding Characteristics
Creation of labels, and encoding of blocks
Target File
Splitting of bsr of the file into basic blocks, replacement of pointers. Write data onto medium.
Record Positioning and Pointer Interpretation Rules
Create file for pointer table
Write file onto medium
Target File on Storage Media
Target File on Storage Media

Figure 2.2 The Use of Descriptions and the Association List in Data Conversion [1]

[1] Reproduced from [Smith 1971, p. 150-152]

syntax and semantic definitions to generate a Data Description Language (DDL) compiler. The DDL compiler formats and translates DDL Data Definition Statements to produce a Data Conversion Processor which in turn converts the source data base to the target data base. This system is illustrated in Figure 2.3.

The design has been completed and the implementation is to be done in PL/1 on an IBM 370/165. However, this is a large project and it remains to be seen if it can also be implemented on computers such as the CDC 6000 series using Fortran without major modifications. Implementation of a system proposing data portability should include the restriction that the system itself is portable. In this way the system does not become another special purpose translation program.

University of Michigan

The ISDOS project has suggested a developmental model for data translation (Fry, et al. 1972). A source converter accepts a source file and its description and produces a normal form. A restructurer accepts this normal form, restructuring specifications, source and target descriptions and produces another normal form. This new normal form and target description are then used as input to the target converter which produces the target file.

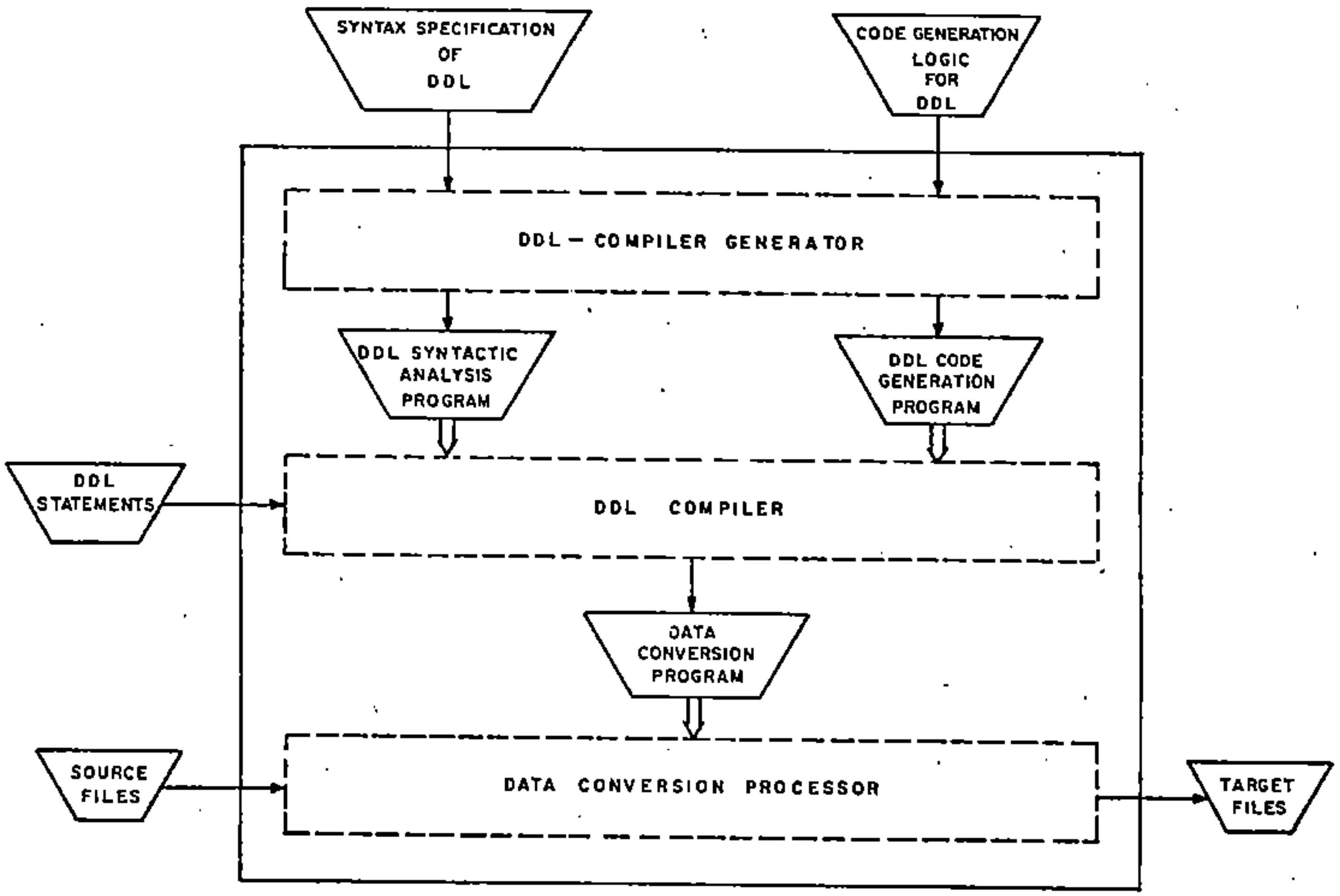

SYNTAX SPECIFICATION OF DDL
CODE GENERATION LOGIC FOR DDL
DDL — COMPILER GENERATOR
DDL SYNTACTIC ANALYSIS PROGRAM
DDL CODE GENERATION PROGRAM
DDL STATEMENTS
DDL COMPILER
DATA CONVERSION PROGRAM
SOURCE FILES
DATA CONVERSION PROCESSOR
TARGET FILES

Figure 2.3 DDL-Processor System [1]

[1] Reproduced from [French, et al. 1971, p. 17]

Specifications for the normal form are not given in the paper. The general model is illustrated in Figure 2.4.

The Data Translation Project is developing a methodology for automation of the file translation process (Sibley 1972). Design specifications have been produced for a prototype data translator which will be the first step in development of a generalized data translator. The implementation will take the NIPS data base management system on the IBM 360 and translate it into files acceptable to the WWDMS data base management system on the HIS 6050. The implementation language on the HIS 6050 will be ANS Fortran with assembly language used where Fortran does not have sufficient capability.

A functional description of the model is as follows. A Stored Data Definition Language (SDDL) Analyzer accepts a description of both target and source inputs written in SDDL. A Table Builder accepts outputs from the SDDL Analyzer and produces input tables for the Translation Modules. A Translation Definition Language (TDL) Analyzer accepts relationships between the attributes and names of items, groups, etc. in the source and target files and produces input tables for the Translation Modules. The Translation Modules then accept input from the source file, SDDL tables, TDL tables and produce a target file (see Figure 2.5).

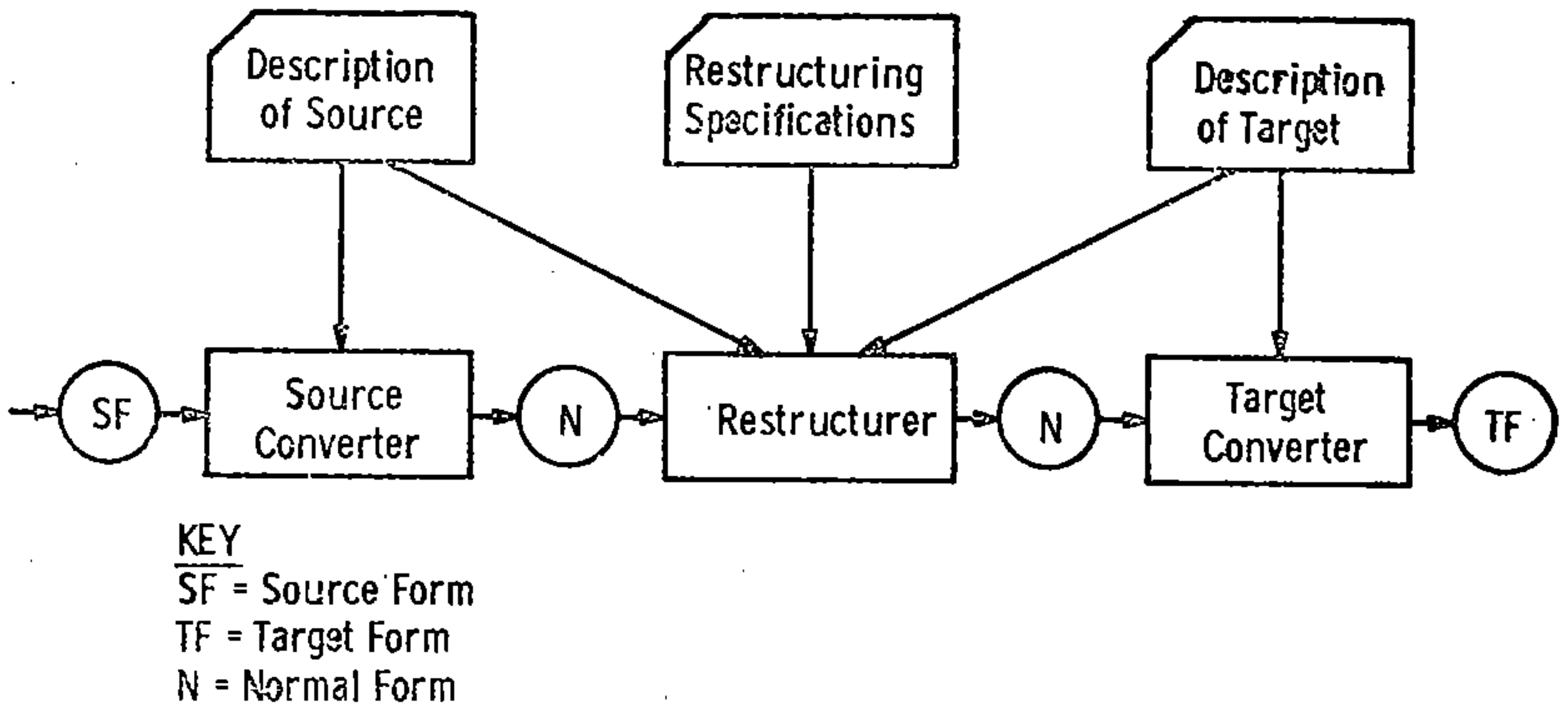


Figure 2.4 The General Model for Data Translation [1]

[1] Reproduced from [Fry, et al. 1972, p. 9]

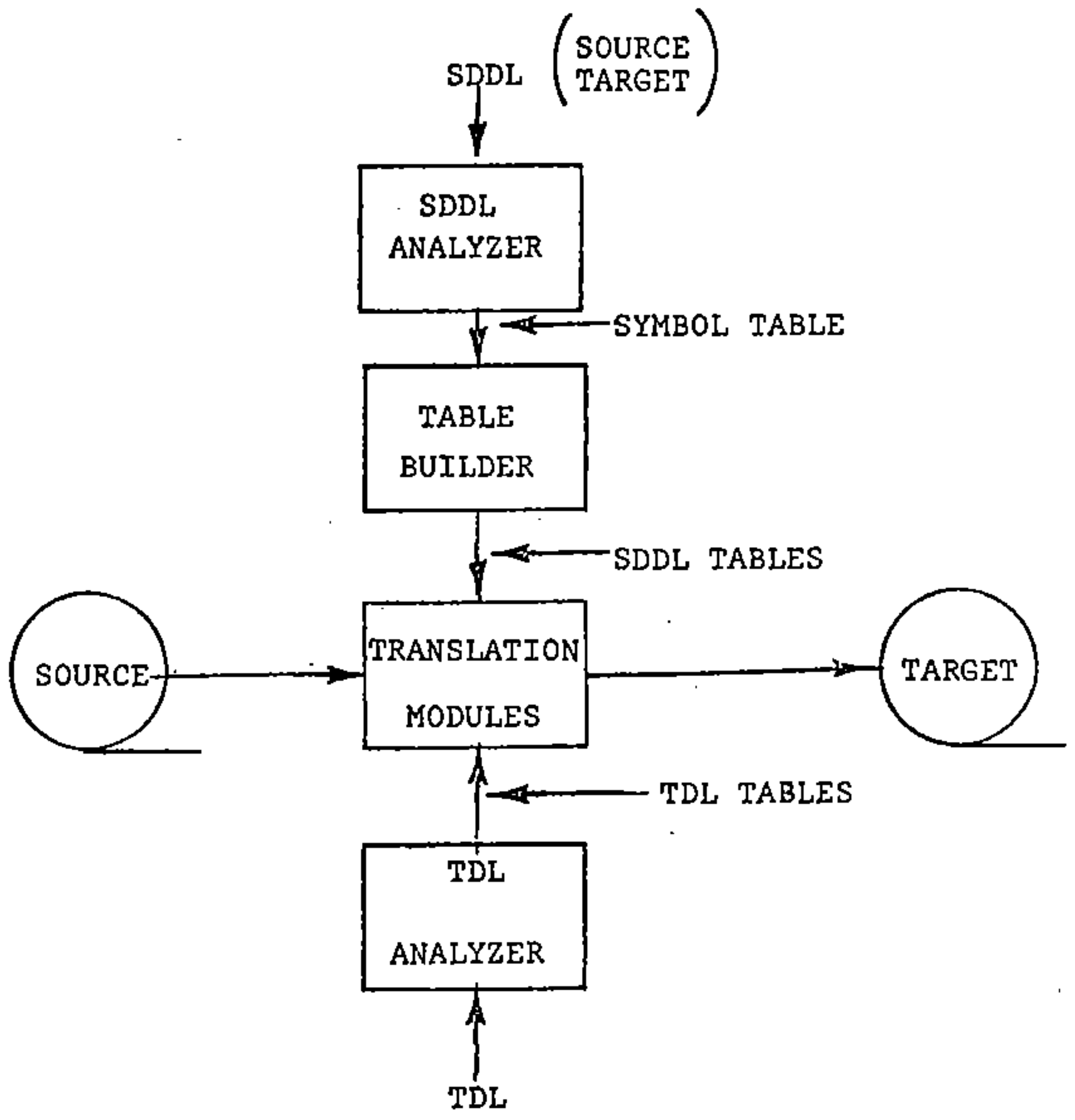


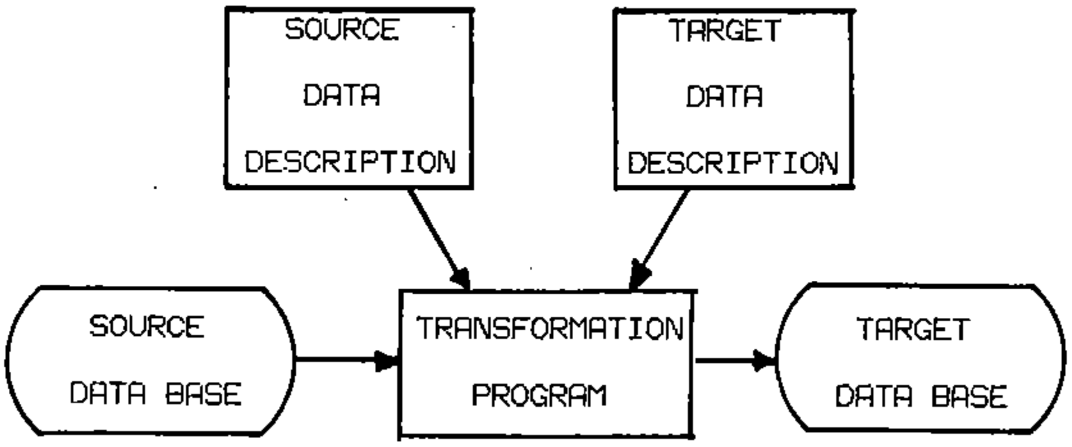
Figure 2.5 Functional Diagram of the Data Translation Project [1]

[1] Reproduced from [Sibley 1972, p. 1.9]

Implementing specific file organizations on specific computers (i.e. NIPS 360 to WWDMS 6050) does not lend itself to portability unless the implementors are extremely careful. Thus, the resulting generalized data translator may not be as general as originally intended.

THE GENERAL PROBLEM

The general problem of data portability can best be expressed by the following diagram.



The <u>source data description</u> is a concise and complete representation of all the data that resides on the source data base, its contents and corresponding relationships.

Both physical and logical constraints exist in the data description.

The target data description is a concise and complete representation of all the data that resides on the target data base, its contents and corresponding relationships. Both physical and logical constraints exist in the data description.

The transformation program consists of an executive that produces a target data base by (1) selecting source fields from the source data base and (2) choosing algorithms (based on source and target data descriptions) to properly transform these source fields to corresponding target fields.

EXAMPLE OF AN ALGORITHM

As an example of one of the algorithms that may exist in the transformation program, consider the algorithm that transforms s single precision floating point to t single precision floating point. Also, let

s = IBM 360

t = CDC 6500

If the transformation program resides on the target computer and d is the target field, the necessary source computer attributes required for this example are:

word size	<1,32>
sign	<1>
characteristic	<2,8>
fraction	<9,32>
characteristic excess	64
exponent base	16
negative number representation	sign-magnitude
fraction radix point	<8.9>

Given the source field, the transformation algorithm is:

1) extract sign	leftmost bit → I1
2) extract characteristic	next leftmost 7 bits → I2
3) extract fraction	rightmost 24 bits → R1
4) extract exponent	I2 - 64 → I3
5) normalize fraction	shift binary point of R1 left 24 bits → R2
6) unsigned answer	R2 * (16 ↑ I3) → R3
7) signed answer	if I I1 = 1 then -R3 → d, otherwise R3 → d

OTHER REQUIRED ATTRIBUTES

Although these previous attributes are required for floating point, additional attributes will be necessary to describe other data types. For example,

1) fixed point will require

    word length
    position of sign
    position of the fraction
    length of the fraction
    negative representation

2) pure graphic transformations will require

    bits per graphic code
    source to target graphic transformation table

3) zoned, packed and numeric decimal will require

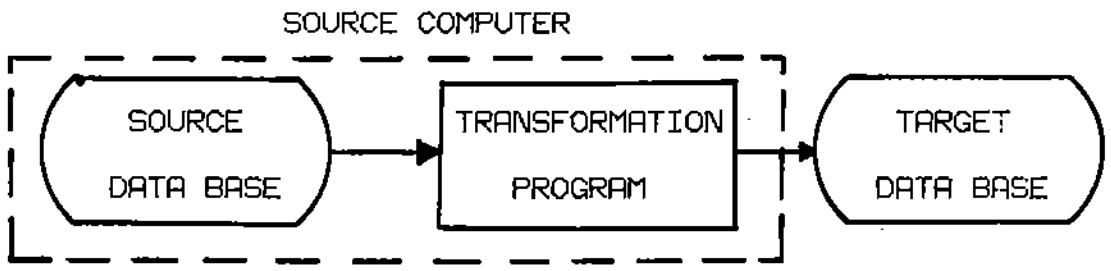
    bits per character
    bits per digit
    source to target graphic transformation tables

For other data types, additional attributes will be required.

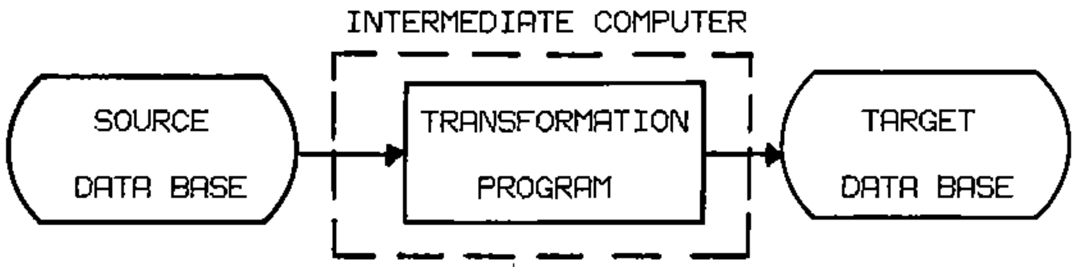
LOCATIONS FOR THE TRANSFORMATION PROGRAM

The transformation program may reside on the source computer, target computer or an intermediate computer. Graphically this can be represented as follows:

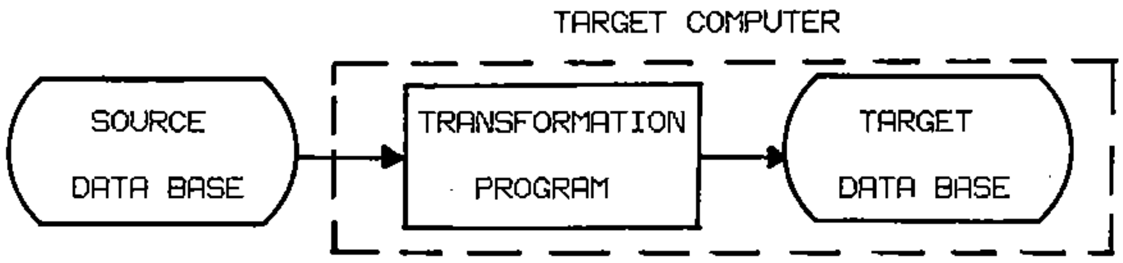
Method 1



Method 2



Method 3



All three methods have their advantages as well as disadvantages. Method 1 assumes that the target data description is well defined when in most cases it is not. It has the distinct advantage however of having the source data description well defined.

Method 2 assumes both the source and target data description are well defined. This method has a major weakness in that for the majority of cases, neither the source data description nor the target data description is well defined.

Method 3 assumes that the source data description is well defined when in most cases it is not. The advantage is that the target data description is well defined.

SYNTHESIS OF PREVIOUS APPROACHES

The approaches discussed earlier in this chapter can be expressed as belonging to one of these three methods. The goals of the groups at University of Pennsylvania and University of Michigan have been towards Method 2. However, the Data Translation Project, the Data Description Language Processor System and the DBTG proposal are being implemented similarly to Method 3. The largest problem to overcome in going from Method 3 to Method 2 is that of being able to have both the source and target data description well defined.

AN ARGUMENT FOR METHOD THREE

Models 1 and 2 require an accurate description of the target data description. This description is very difficult to obtain when even the recipient of the data base may not be able to provide this information. Only Method 3 does not require advance knowledge of the target computer. Method 3 has met with limited success in the form of one-shot transformation programs which have been developed iteratively. The iterative process is due primarily to the lack of information about the source data base originally. Thus, if success is not met, additional information is gathered, the program modified and another attempt or iteration is made.

Therefore, if it is possible to (1) construct a portable transformation program that can exist on the target computer (2) reduce the number of iterations necessary to transform the source data base and (3) derive a workable data description that allows a user to easily redefine his data base as more relevant information is discovered, we are on the way towards providing data base portability.

As an approach to constructing a portable transformation program, let us examine the most common portable medium today (i.e. magnetic tape). Before the problem of portable magnetic tape is dismissed as being trivial, realize that the problem has existed for almost two decades and the general problem has yet to be solved. There is considerable difficulty in reading "strange magnetic tapes" at most computer installations today.

Having established that the first step to solving the problem of data portability is in the area of magnetic tapes, some problems that exist with magnetic tapes are examined in the next chapter.

CHAPTER 3

PROBLEMS ARISING IN MAGNETIC TAPE PORTABILITY

GENERAL COMMENTS

The problems arising in magnetic tape portability lie primarily in the areas of standardization, data compression, physical constraints, graphic codes and creating acceptable source tapes. These problems are discussed in general followed by specific examples that have occurred in 20 test cases.

STANDARDS

Standards for recorded magnetic tape for information interchange have been specified by the American National Standards Institute (ANSI 1967a, ANSI 1970). A standard for internal magnetic tape labels for information interchange also has been published (ANSI 1967b). However, many tapes do not conform to the standards. Computer companies also publish manuals describing their internal labels for specific operating systems (IBM 1972) as well as standards for information interchange between different operating systems (IBM 1970). In addition, the Department of Commerce has published some broad guidelines for describing information interchange formats which also are applicable to magnetic tapes (FIPS PUB 20 1972).

A typical user when sending a magnetic tape, uses utility routines to copy his program or data from mass storage to magnetic tape. Unfortunately these utility routines may copy the data directly in the form used by the operating system. In many operating systems, as an efficiency measure, physical records on magnetic tape conform to tracks or fractions of tracks on discs or drums. On other computers, the length of a physical record is determined by the size and capabilities of a peripheral processor. Therefore, a track or a fraction of a track on disc or a core load for a peripheral processor may be mapped to a physical record on tape. This results in a tape that is easily read only by the source computer. These deviations from the specified standards are due primarily to the increasing amount of file handling irrespective of the storage device.

Problems also occur when tapes are blocked. Blocking allows several logical records to be contained in one physical record thus eliminating excessive end of record gaps that occupy useable space on tape. Blocking is acceptable if the physical record size is acceptable to the target computer. Often, due to memory or buffer size limitations, standard utility routines are not available to read large physical records from the tape. Hence, the tape must first be unblocked using non-standard routines so that it may run on the target computer.

Some operating systems require special formatted tapes. One example is OS 370. Tapes may be labeled or unlabeled. If they are labeled, the label may contain no more than 120 8-bit bytes. Tapes may have variable length physical records but the length of the variable physical record must be in the first four bytes of each physical record. This implies that if a tape generated on a UNIVAC 1108 contains 126 bytes in the first physical record and 1008 bytes per physical record after that, the job control language (JCL) will not allow the tape to be read unless you specify the tape as "unknown".

Standards for one computer system are not necessarily standards on another computer system. Enforcing national standards on all users is a gigantic task.

COMPRESSED DATA

Data is compressed in order to save space. Trailing blanks may be deleted by replacing them with a set of characters indicating an end of line condition. One example is the tape generated by the MACE operating system on the CDC 6500. Trailing blanks are deleted from line images with a special end of line condition, 12 binary zeroes in the rightmost 12 bits of the 60 bit word, specifying the end of the line. These trailing blanks are replaced when the compressed line image is read by the system to be used as data. Another technique is to have a set of characters or binary string indicating its compressed length. An example of this may be found on the Univac 1108 where the word preceeding the line image contains, among other things, the number of 36 bit words that follow.

A more complicated technique involves deleting all occurrences of three or more blanks. An example of this is XFILES under the MACE operating system on the CDC 6500 where the first blank is replaced with binary zeroes and the second blank is replaced with a binary counter indicating the number of blanks deleted. This technique can also be used to replace multiple occurrences of other characters by replacing them with a special symbol, or a count accompanied by a special symbol. Increasingly complex techniques involve compressing bits rather than characters by replacement of special strings of bits. More sophisticated techniques used in compressing large business files are summarized in Ruth and Kreutzer (1972).

There are many ways to compress data and it is not possible at this time to develop a general decompression algorithm. A decompression algorithm unpacks the data from a packed form into a useable form. Since decompression routines within the operating system are transparent to the user, it is possible for him to inadvertently create a compressed tape to send to another computer installation. If this occurs, it is his responsibility to also send the decompression algorithm. The target computer installation can then apply the algorithm and decompress the tape.

PHYSICAL CONSTRAINTS

Reconstructing data on damaged tape is very difficult and if the damage to the tape is severe enough, the user must request that a new tape be generated again at the source site. Tape drives can be out of adjustment when they are recording. This may result by the tape head not being perpendicular to the magnetic tape (skewing), or the recorded density may not be within the allowable tolerances.

Also, the tape itself may have creases in it, the oxide may be coming off, it may be dirty, old, etc.

Another problem is receiving tapes that are readable only on specific tape drives. For example, it is not possible to read a 1600 bpi tape on a single density 800 bpi tape drive. Likewise, 9-track tapes are not readable on 7-track tape drives. In this case, the user must either have it copied to a useable density with the correct number of tracks or he must request a new source tape that is physically compatible.

GRAPHIC CODES

Graphic codes are universally misunderstood. The user sees a graphic symbol as it is represented on a listing or a display tube. Graphic symbol sets are in turn controlled by the slugs on a print chain, type bars or matrices on terminals and the available graphics on a display tube. Since these symbols are what the user sees and uses, he does not generally care about their internal representation in memory. As a direct result, he believes that the character set that he uses is rather universally known when in fact it is not. As an example, Binary Coded Decimal (BCD) codes are often thought to be the same on different computers. This is not true. In particular, the graphic symbols for CDC 6000 series external BCD are not the same as for the BCD used on the IBM 360 series (see Tables 3.1 and 3.7).

The three major graphic codes appearing on magnetic tape today are ASCII, EBCDIC, and various versions of BCD. The BCD code (6 bits) was developed from a binary representation of the existing hollerith punch card code (Clamons 1971). EBCDIC (8 bits) was developed by IBM to provide an Extended Binary Coded Decimal Interchange Code that was more than 6 bits long. It is used as an internal and external code on the IBM 360/370 series which is 8-bit oriented. ASCII (American Standard Code for Information Interchange) is a 7-bit code based on the ISO (International Standards Organization) recommendations (FIPS PUB 1 1968).

Let us now examine each of the above graphic codes along with some others that are in common use.

When BCD codes are used on IBM computers, they are generally of two forms. These forms are BCDIC (principle) and BCDIC (alternate).

BCDIC (PRINCIPLE)

The Binary Coded Decimal Interchange Code (BCDIC) is an outgrowth of the BCD codes on the early IBM machines. BCDIC (principle) contains 64 characters and is commonly referred to as the commercial character set. It differs from the scientific character set (see BCD alternate below) in that it allows the 5 special symbols: ampersand, percent, number sign, commercial at and lozenge (see Table 3.1)

BCDIC (ALTERNATE)

This code is similar to BCDIC (principle) except for 5 scientific symbols which replace the commercial symbols. They are: opening parenthesis, plus sign, closing parenthesis, apostrophe and equal sign (see Table 3.1).

The two BCDIC codes are not universal across all computers. In fact, BCD Basic (see below) seems to be the most universal BCD code with BCD Basic + 7 (see below) being applicable the majority of the time.

BCD BASIC

Binary Coded Decimal (BCD) Basic is a 6-bit code consisting of the letters A through Z and the numerics 0 through 9. The majority of BCD codes contain the same numeric representation for these 36 graphics. (see Table 3.2)

Table 3.1   Standard BCD Interchange Code and Graphics [1]

COLLATING NUMBER*	ASSIGNED GRAPHICS	PRINCIPLE GRAPHIC NAME	ALTERNATE GRAPHIC NAME	CARD CODE**			SEVEN-TRACK MAGNETIC TAPE BCD CODE (EVEN PARITY)						
							C	B	A	8	4	2	1
00	BLANK	Blank (Space)		NO PUNCHES			NO BITS***						
01	.	Period		12	3	8	C	B	A	8		2	1
02	⌑)	Lozenge	Right Parenthesis	12	4	8		B	A	8	4		
03	[	Left Bracket		12	5	8	C	B	A	8	4		1
04	<	Less-Than Sign		12	6	8	C	B	A	8	4	2	
05	≢	Group Mark		12	7	8		B	A	8	4	2	1
06	& +	Ampersand	Plus	12				B	A				
07	$	Dollar Sign		11	3	8		B		8		2	1
08	*	Asterisk		11	4	8	C	B		8	4		
09	]	Right Bracket		11	5	8		B		8	4		1
10	;	Semi-Colon		11	6	8		B		8	4	2	
11	Δ	Mode Change		11	7	8	C	B		8	4	2	1
12	-	Minus Sign, Hyphen		11			C	B					
13	/	Slash		0	1				A				1
14	,	Comma		0	3	8			A	8		2	1
15	% (	Per Cent Sign	Left Parenthesis	0	4	8	C		A	8	4		
16	V	Word Separator		0	5	8			A	8	4		1
17	\	Backslash		0	6	8			A	8	4	2	
18	⧧	Segment Mark		0	7	8	C		A	8	4	2	1
19 ***	b̸	Substitute Blank			2	8	C		A				
20	# =	Number Sign	Equal Sign		3	8	C			8		2	1
21	@ '	At Sign	Prime, Apostrophe		4	8				8	4		
22	:	Colon			5	8	C			8	4		1
23	>	Greater - Than Sign			6	8	C			8	4	2	
24	√	Tape Mark (Radical)			7	8				8	4	2	1
25	?	Question Mark		12	0			B	A	8		2	
26	A			12	1		C	B	A				1
27	B			12	2		C	B	A			2	
28	C			12	3			B	A			2	1
29	D			12	4		C	B	A		4		
30	E			12	5			B	A		4		1
31	F			12	6			B	A		4	2	
32	G			12	7		C	B	A		4	2	1
33	H			12	8		C	B	A	8			
34	I			12	9			B	A	8			1
35	!	Exclamation Point		11	0		C	B		8		2	
36	J			11	1			B					1
37	K			11	2			B				2	
38	L			11	3		C	B				2	1
39	M			11	4			B			4		
40	N			11	5		C	B			4		1
41	O			11	6		C	B			4	2	
42	P			11	7			B			4	2	1
43	Q			11	8			B		8			
44	R			11	9		C	B		8			1
45	‡	Record Mark		0	2	8	C		A	8		2	
46	S			0	2				A			2	
47	T			0	3		C		A			2	1
48	U			0	4				A		4		
49	V			0	5		C		A		4		1
50	W			0	6		C		A		4	2	
51	X			0	7				A		4	2	1
52	Y			0	8				A	8			
53	Z			0	9		C		A	8			1
54	0				0					8		2	
55	1				1		C						1
56	2				2		C					2	
57	3				3							2	1
58	4				4		C				4		
59	5				5						4		1
60	6				6						4	2	
61	7				7		C				4	2	1
62	8				8		C			8			
63	9				9					8			1

[1] Reproduced from ["IBM Systems/360 Component Description, 2400-Series Magnetic Tape Units, 2803/2804 Tape Control and 2816 Switching Unit, Model 1", GA22-6866-5, Sixth Edition, November 1970, p. 21]

Table 3.2 BCD Basic and BCD Basic+7 Character Codes

BCD BASIC	CODE	BCD BASIC + 7
A	61	A
B	62	B
C	63	C
D	64	D
E	65	E
F	66	F
G	67	G
H	70	H
I	71	I
J	41	J
K	42	K
L	43	L
M	44	M
N	45	N
O	46	O
P	47	P
Q	50	Q
R	51	R
S	22	S
T	23	T
U	24	U
V	25	V
W	26	W
X	27	X
Y	30	Y
Z	31	Z
0	12	0
1	01	1
2	02	2
3	03	3
4	04	4
5	05	5
6	06	6
7	07	7
8	10	8
9	11	9
	20	BLANK
	33	COMMA
	73	PERIOD
	53	DOLLAR SIGN
	54	ASTERISK
	40	MINUS
	21	SLANT

BCD BASIC + 7

Binary Coded Decimal (BCD) Basic + 7 is a 6-bit code consisting of the letters A through Z, the numerics 0 through 9, and the special symbols: blank, comma, period, slash, asterisk, minus and dollar sign. It has 7 more graphic symbols in addition to those in BCD Basic (see table 3.2). These 43 graphic symbols have been found to have common numeric representations in 15 BCD codes by DUAL labs who are responsible for distributing the 1970 census tapes (USDC 1970). It should be noted that BCD Basic + 7 does not contain all of the operators necessary in Basic Fortran.

There are also two standard interchange codes, ANSI and ISO, that have been developed by their respective standards groups.

ASCII, ANSI or ASCII-7

The American National Standards Institute (ANSI) has established this code as the American Standard Code for Information Interchange (ASCII). It is a 7-bit code with all 128 characters being defined in FIPS PUB 1(1968). See Table 3.3 for the code. It was hoped that this graphic code would become a standard. It is not as universally used as was originally intended by its developers

Table 3.3 USA Standard Code for Information Interchange [1]

Bits b7 b6 b5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b4	b3	b2	b1	COLUMN → / ROW ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	–	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	___	o	DEL

[1] Reproduced from [FIPS PUB 1 1968, p. 6]

due to EBCDIC (Clamons 1971). Also, 128 characters do not allow all of the current existing symbols to be represented.

ISO

The International Standards Organization (ISO) has established this code (Ross 1964). It is very similar to 7-bit ASCII and is the standard interchange code to be used by countries outside of the U.S.A. (see Table 3.4).

IBM has also established codes that are de-facto standards. They are EBCDIC and ASCII-8.

EBCDIC

Extended Binary Coded Decimal Interchange Code (EBCDIC) is the 8-bit code used as the internal computer code by the IBM System 360/370 (see Table 3.5). It is also used as the normal external tape code.

ASCII-8

ASCII-7 was extended by IBM to an 8-bit code resulting in ASCII-8 (see Table 3.6). Excluding the transmission control graphics, it corresponds closely to ASCII-7 with one additional bit. Only 128 characters are defined.

Table 3.4 I.S.O. 7-Bit Character Code Table [1]

b$_7$	b$_6$	b$_5$												
							0	0	0	0	1	1	1	1
							0	0	1	1	0	0	1	1
							0	1	0	1	0	1	0	1
			b$_4$	b$_3$	b$_2$	b$_1$	0	1	2	3	4	5	6	7
			0	0	0	0	0	(TC$_0$) Null	(TC$_7$) DLE	Space	0	—	P	(@) [3]	p
			0	0	0	1	1	(TC$_1$) SOH	DC$_1$	!	1	A	Q	a	q
			0	0	1	0	2	(TC$_2$) STX	DC$_2$	" [8]	2	B	R	b	r
			0	0	1	1	3	(TC$_3$) ETX	DC$_3$	$\neq$ [6]	3	C	S	c	s
			0	1	0	0	4	(TC$_4$) EOT	DC$_4$ (Stop)	CS$_1$ [5]	4	D	T	d	t
			0	1	0	1	5	(TC$_5$) Enq	(TC$_8$) Nack	%	5	E	U	e	u
			0	1	1	0	6	(TC$_6$) Ack	(TC$_9$) Sync	&	6	F	V	f	v
			0	1	1	1	7	Bell	(TC$_{10}$) ETB	' [8]	7	G	W	g	w
			1	0	0	0	8	FE$_0$ (BS)	Cancel	(	8	H	X	h	x
			1	0	0	1	9	FE$_1$ (HT)	EM	)	9	I	Y	i	y
			1	0	1	0	10	FE$_2$ (LF) [1]	SS	*	: [2]	J	Z	j	z
			1	0	1	1	11	FE$_3$ (VT)	Escape	+	; [2]	K	([) [3]	k	[3]
			1	1	0	0	12	FE$_4$ (FF)	IS$_4$ (FS)	,	<	L	(CS$_2$) [3]	l	[3]
			1	1	0	1	13	FE$_5$ (CR) [1]	IS$_3$ (GS)	–	=	M	(]) [3]	m	[3]
			1	1	1	0	14	SO	IS$_2$ (RS)	.	>	N	^ [7] [8]	n	[3]
			1	1	1	1	15	SI	IS$_1$ (US)	/	?	O	_	o	Delete

Explanatory notes about the 7-bit code table

1. The controls CR and LF are intended for printer equipment which requires separate combinations to return the carriage and to feed a line.
   For equipment which uses a single combination (called New Line) for combined carriage return and line feed operation, NL will be coded at FE$_2$. The use of New Line requires agreement between the sender and the recipient of the data.
2. If 10 and 11 as single characters are needed (for example, for Sterling currency subdivision), they should take the place of "colon" (:) and "semi-colon" (;) respectively.
3. "Reserved for National Use." These positions are primarily intended for alphabetic extensions. If they are not required for that purpose, they may be used for symbols and the recommended choice as shown in parentheses.
5. "Reserved for National Use." A currency sign will be assigned to this position.
6. The number sign ($\neq$) in position 2/3 may have an alternate graphical representation (N°).
7. It is acceptable to represent tilde (~) by circumflex (^) for international interchange of information in Spanish and Portuguese. In Spanish and Portugese speaking countries the tilde may replace the circumflex in position 5/14.
8. The graphics in positions 2/2, 2/7, 5/14 have the significance of quotation mark, apostrophe and upwards arrow, respectively; however, these characters take on the significance of diaeresis, acute accent and circumflex diacritical signs when they follow the Backspace code.

[1] Reproduced from [Ross 1964, p. 199]

Table 3.5 Extended Binary Coded Decimal Interchange Code [1]

Bit Positions 4,5	6,7	00				01				10				11				Bit Positions ←0,1
		00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11	←2,3
00	00	NUL		DS		SP	&	-									0	
	01			SOS				/		a	j			A	J		1	
	10			FS						b	k	s		B	K	S	2	
	11		TM							c	l	t		C	L	T	3	
01	00	PF	RES	BYP	PN					d	m	u		D	M	U	4	
	01	HT	NL	LF	RS					e	n	v		E	N	V	5	
	10	LC	BS	EOB	UC					f	o	w		F	O	W	6	
	11	DEL	IL	PRE	EOT					g	p	x		G	P	X	7	
10	00									h	q	y		H	Q	Y	8	
	01									i	r	z		I	R	Z	9	
	10		CC	SM		¢	!		:									
	11					.	$	,	#									
11	00					<	*	%	@									
	01					(	)	–	'									
	10					+	;	>	=									
	11					|	¬	?	"									

[1] Reproduced from ["IBM Systems/360 Component Description, 2400-Series Magnetic Tape Units, 2803/2804 Tape Control and 2816 Switching Unit, Model 1", GA22-6866-5, Sixth Edition, November 1970, p. 18]

Table 3.6 Eight-Bit Representation for USASCII-8 [1]

Bit Positions 4,3	2,1	00				01				10				11			
		00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
00	00	NUL	DLE			SP	0					@	P			`	p
	01	SOH	DC1			!	1					A	Q			a	q
	10	STX	DC2			"	2					B	R			b	r
	11	ETX	DC3			#	3					C	S			c	s
01	00	EOT	DC4			$	4					D	T			d	t
	01	ENQ	NAK			%	5					E	U			e	u
	10	ACK	SYN			&	6					F	V			f	v
	11	BEL	ETB			'	7					G	W			g	w
10	00	BS	CAN			(	8					H	X			h	x
	01	HT	EM			)	9					I	Y			i	y
	10	LF	SS			*	:					J	Z			j	z
	11	VT	ESC			+	;					K	[			k	{
11	00	FF	FS			,	<					L	\			l	|
	01	CR	GS			–	=					M	]			m	}
	10	SO	RS			.	>					N	^			n	~
	11	SI	US			/	?					O	—			o	DEL

Bit Positions ←8,7 ←6,5

[1] Reproduced from ["IBM Systems/360 Component Description, 2400-Series Magnetic Tape Units, 2803/2804 Tape Control and 2816 Switching Unit, Model 1", GA22-6866-5, Sixth Edition, November 1970, p. 18]

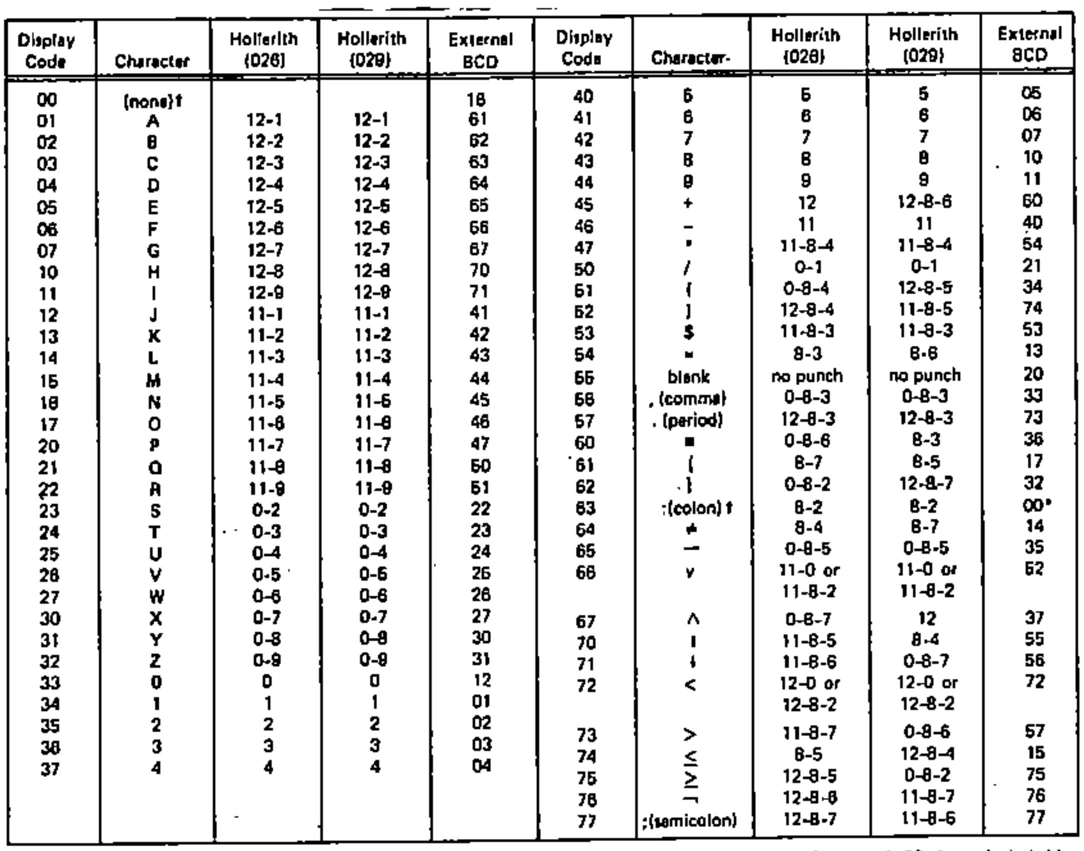
On many computers, reference is made to external BCD and internal BCD. External BCD is a graphic code that exists on an external media such as magnetic tape. When this code is read into main memory from magnetic tape, a mapping takes place, usually by the hardware, to transform the external BCD to internal BCD. Among other things, this is largely due to not being able to represent a zero as a true zero on even parity tape. A BCD zero is normally recorded on magnetic tape as an octal 12.

Generally if a specific computer has an internal code different from BCD a one to one relationship exists when the number of bits in each of the codes is the same. As an example of various graphic codes that exist within one computer, consider the CDC 6500. There are basically 3 codes. (1) DISPLAY which is the internal code of the computer, (2) external BCD, a code that can exist on mass storage devices and (3) internal BCD, a code that is directly related to external BCD except that it resides in the computer (see Table 3.7).

DISPLAY (CDC 6500)

This is the 6-bit internal machine code used on the CDC 6500 as well as on other CDC 6000 series machines. (see Table 3.7).

Table 3.7 CDC 63-Character Set [1]



Display Code	Character	Hollerith (026)	Hollerith (029)	External BCD	Display Code	Character	Hollerith (026)	Hollerith (029)	External BCD
00	(none)†			16	40	5	5	5	05
01	A	12-1	12-1	61	41	6	6	6	06
02	B	12-2	12-2	62	42	7	7	7	07
03	C	12-3	12-3	63	43	8	8	8	10
04	D	12-4	12-4	64	44	9	9	9	11
05	E	12-5	12-5	65	45	+	12	12-8-6	60
06	F	12-6	12-6	66	46	–	11	11	40
07	G	12-7	12-7	67	47	*	11-8-4	11-8-4	54
10	H	12-8	12-8	70	50	/	0-1	0-1	21
11	I	12-9	12-9	71	51	(	0-8-4	12-8-5	34
12	J	11-1	11-1	41	52	)	12-8-4	11-8-5	74
13	K	11-2	11-2	42	53	$	11-8-3	11-8-3	53
14	L	11-3	11-3	43	54	=	8-3	8-6	13
15	M	11-4	11-4	44	55	blank	no punch	no punch	20
16	N	11-5	11-5	45	56	, (comma)	0-8-3	0-8-3	33
17	O	11-6	11-6	46	57	. (period)	12-8-3	12-8-3	73
20	P	11-7	11-7	47	60	≡	0-8-6	8-3	36
21	Q	11-8	11-8	50	61	[	8-7	8-5	17
22	R	11-9	11-9	51	62	]	0-8-2	12-8-7	32
23	S	0-2	0-2	22	63	:(colon) †	8-2	8-2	00*
24	T	0-3	0-3	23	64	≠	8-4	8-7	14
25	U	0-4	0-4	24	65	→	0-8-5	0-8-5	35
26	V	0-5	0-5	25	66	∨	11-0 or 11-8-2	11-0 or 11-8-2	52
27	W	0-6	0-6	26	67	∧	0-8-7	12	37
30	X	0-7	0-7	27	70	↑	11-8-5	8-4	55
31	Y	0-8	0-8	30	71	↓	11-8-6	0-8-7	56
32	Z	0-9	0-9	31	72	<	12-0 or 12-8-2	12-0 or 12-8-2	72
33	0	0	0	12	73	>	11-8-7	0-8-6	57
34	1	1	1	01	74	≤	8-5	12-8-4	15
35	2	2	2	02	75	≥	12-8-5	0-8-2	75
36	3	3	3	03	76	¬	12-8-6	11-8-7	76
37	4	4	4	04	77	;(semicolon)	12-8-7	11-8-6	77

†When the 63-Character Set is used, the punch code 8-2 is associated with display code 63, the colon. Display code 00_8 is not included in the 63-Character Set and is not associated with any card punch. The 8-6 card punch (026 keypunch) and the 0-8-4 card punch (029 keypunch) in the 63-Character Set are treated as blank on input.

*Since 00 cannot be represented on magnetic tape, it is converted to BCD 12. On input, it will be translated to display code 33 (number zero).

[1] Reproduced from ["Control Data 6000 Systems Fortran Reference Manual, 6000 Version 2.3, No. 60174900E", April 23, 1971, Appendix A p. 4]

EXTERNAL BCD (CDC 6500)

This code is used when a user requests that his data be read or written in external BCD. Basic BCD is a sub set of this code. (see Table 3.7).

INTERNAL BCD (CDC 6500)

This is the internal representation of BCD after its conversion from external BCD. (see Table 3.7).

There are also several other internal computer codes such as FIELDATA used on the UNIVAC 1100 series, Internal BCD on the Honeywell 6000 series and BCL on the Burroughs 6500. These may be found respectively in Tables 3.8, 3.9 and 3.10.

ACCEPTABLE SOURCE TAPES

There are several problems that arise when tapes are generated on the source computer. For example, consider the IBM 360 and 370 series. When generating a 7-track tape, four options can be used.

a) TRTCH = T - The 8-bit EBCDIC code is translated to the 6-bit BCDIC code. Since the EBCDIC code has more than 64 states and the BCDIC code has only 64 possible

Table 3.8 1108 Character Codes [1]

INT. Octal	BCD Tape Octal	Punch Code	1108 On-Line Printer	INT. Octal	BCD Tape Octal	Punch Code	1108 On-Line Printer
00	17	7-8	@	40	74	12-4-8	)
01	75	12-5-8	[[1]	41	40	11	-
02	55	11-5-8	] [1]	42	60	12	+
03	77	12-7-8	#	43	76	12-6-8	<
04	57	11-7-8	Δ [1]	44	13	3-8	=
05	20	(BLANK)	(Space)	45	16	6-8	>
06	61	12-1	A	46	00	2-8	& [2]
07	62	12-2	B	47	53	11-3-8	$
10	63	12-3	C	50	54	11-4-8	*
11	64	12-4	D	51	34	0-4-8	(
12	65	12-5	E	52	35	0-5-8	%
13	66	12-6	F	53	15	5-8	:
14	67	12-7	G	54	72	12-0	?
15	70	12-8	H	55	52	11-0	!
16	71	12-9	I	56	33	0-3-8	, [1]
17	41	11-1	J	57	36	0-6-8	\
20	42	11-2	K	60	12	0	0
21	43	12-3	L	61	01	1	1
22	44	11-4	M	62	02	2	2
23	45	11-5	N	63	03	3	3
24	46	11-6	O	64	04	4	4
25	47	11-7	P	65	05	5	5
26	50	11-8	Q	66	06	6	6
27	51	11-9	R	67	07	7	7
30	22	0-2	S	70	10	8	8
31	23	0-3	T	71	11	9	9
32	24	0-4	U	72	14	4-8	'
33	25	0-5	V	73	56	11-6-8	;
34	26	0-6	W	74	21	0-1	/
35	27	0-7	X	75	73	12-3-8	.
36	30	0-8	Y	76	37	0-7-8	π [1] [2]
37	31	0-9	Z	77	32	0-2-8	(Space)[1]

All other combinations of punches are illegal to the 1108 on-line card reader and cause a reader stop. All combinations of punches are accepted by the 9200-9300 and converted to some character. Most "illegal" punches are converted to blanks (octal 05).

NOTES

1. These characters are different on the 9200 or 9300 printers.

2. This character cannot be written on even parity (BCD) tape; if it is present in an output record, the physical record is truncated with the previous frame (character).

3. The character is input from the 9200 or 9300 card reader and transmitted to the 1108 correctly. The character can be output punched. If the character is in a print line, it is converted to the > before transmission from the 1108 to 9200 or 9300 printer. Also this character is temporarily illegal on the 1004.

4. This character prints as a space on the 1108 printer and truncates the print line.

[1] Reproduced from [University of Wisconsin internal document, September 10, 1969, p. B-4, B-5]

Table 3.9 GE-625/635 Standard Character Set [1]

Standard Character Set	GE-Internal Machine Code	Octal Code	Hollerith Card Code	Standard Character Set	GE-Internal Machine Code	Octal Code	Hollerith Card Code
0	00 0000	00	0	↑	10 0000	40	11-0
1	00 0001	01	1	J	10 0001	41	11-1
2	00 0010	02	2	K	10 0010	42	11-2
3	00 0011	03	3	L	10 0011	43	11-3
4	00 0100	04	4	M	10 0100	44	11-4
5	00 0101	05	5	N	10 0101	45	11-5
6	00 0110	06	6	O	10 0110	46	11-6
7	00 0111	07	7	P	10 0111	47	11-7
8	00 1000	10	8	Q	10 1000	50	11-8
9	00 1001	11	9	R	10 1001	51	11-9
[	00 1010	12	2-8	-	10 1010	52	11
#	00 1011	13	3-8	$	10 1011	53	11-3-8
@	00 1100	14	4-8	*	10 1100	54	11-4-8
:	00 1101	15	5-8	)	10 1101	55	11-5-8
>	00 1110	16	6-8	;	10 1110	56	11-6-8
?	00 1111	17	7-8	'	10 1111	57	11-7-8
ƀ	01 0000	20	(blank)	+	11 0000	60	12-0
A	01 0001	21	12-1	/	11 0001	61	0-1
B	01 0010	22	12-2	S	11 0010	62	0-2
C	01 0011	23	12-3	T	11 0011	63	0-3
D	01 0100	24	12-4	U	11 0100	64	0-4
E	01 0101	25	12-5	V	11 0101	65	0-5
F	01 0110	26	12-6	W	11 0110	66	0-6
G	01 0111	27	12-7	X	11 0111	67	0-7
H	01 1000	30	12-8	Y	11 1000	70	0-8
I	01 1001	31	12-9	Z	11 1001	71	0-9
&	01 1010	32	12	←	11 1010	72	0-2-8
.	01 1011	33	12-3-8	,	11 1011	73	0-3-8
]	01 1100	34	12-4-8	%	11 1100	74	0-4-8
(	01 1101	35	12-5-8	=	11 1101	75	0-5-8
<	01 1110	36	12-6-8	"	11 1110	76	0-6-8
\	01 1111	37	12-7-8	!	11 1111	77	0-7-8

[1] Reproduced from ["GE-625/635 Programming Reference Manual CPB-1004C", August 1965, Appendix F]

Table 3.10 B6500 Data Representation ¹

GRAPHIC	BCL EXTERNAL	BCL INTERNAL	EBCDIC INTERNAL	HEXADECIMAL GRAPHIC
Blank	01 0000	11 0000	0100 0000	40
.	11 1011	01 1010	0100 1011	4B
[	11 1100	01 1011	0100 1010	4A
(	11 1101	01 1101	0100 1101	4D
<	11 1110	01 1110	0100 1100	4C
←	11 1111	01 1111	0100 1111	4F
&	11 0000	01 1100	0101 0000	50
$	10 1010	10 1010	0101 1011	5B
*	10 1100	10 1011	0101 1100	5C
)	10 1101	10 1101	0101 1101	5D
;	10 1110	10 1110	0101 1110	5E
≤	10 1111	10 1111	0101 1111	5F
-	10 0000	10 1100	0110 0000	60
/	01 0001	11 0001	0110 0001	61
,	01 1011	11 1010	0110 1011	6B
%	01 1100	11 1011	0110 1100	6C
=	01 1110	11 1101	0111 1110	7E
]	01 1110	11 1110	0101 1010	5A
"	01 1111	11 1111	0111 1111	7F
#	00 1011	00 1010	0111 1011	7B
@	00 1100	00 1011	0111 1100	7C
:	00 1101	00 1101	0111 1010	7A
>	00 1110	00 1110	0110 1110	6E
≥	00 1111	00 1111	0111 1101	7D

GRAPHIC	BCL EXTERNAL	BCL INTERNAL	EBCDIC INTERNAL	HEXADECIMAL GRAPHIC
+	11 1010	01 0000	1100 0000	C0
A	11 0001	01 0001	1100 0001	C1
B	11 0010	01 0010	1100 0010	C2
C	11 0011	01 0011	1100 0011	C3
D	11 0100	01 0100	1100 0100	C4
E	11 0101	01 0101	1100 0101	C5
F	11 0110	01 0110	1100 0110	C6
G	11 0111	01 0111	1100 0111	C7
H	11 1000	01 1000	1100 1000	C8
I	11 1001	01 1001	1100 1001	C9
x (Mult.)	10 1010	10 0000	1101 0000	D0
J	10 0001	10 0001	1101 0001	D1
K	10 0010	10 0010	1101 0010	D2
L	10 0011	10 0011	1101 0011	D3
M	10 0100	10 0100	1101 0100	D4
N	10 0101	10 0101	1101 0101	D5
O	10 0110	10 0110	1101 0110	D6
P	10 0111	10 0111	1101 0111	D7
Q	10 1000	10 1000	1101 1000	D8
R	10 1001	10 1001	1101 1001	D9
≠	01 1010	11 1100	0110 1101	6D
S	01 0010	11 0010	1110 0010	E2
T	01 0011	11 0011	1110 0011	E3
U	01 0100	11 0100	1110 0100	E4
V	01 0101	11 0101	1110 0101	E5
W	01 0110	11 0110	1110 0110	E6
X	01 0111	11 0111	1110 0111	E7
Y	01 1000	11 1000	1110 1000	E8
Z	01 1001	11 1001	1110 1001	E9

Table 3.10, cont.

GRAPHIC	BCL EXTERNAL	BCL INTERNAL	EBCDIC INTERNAL	HEXADECIMAL GRAPHIC
0	00 1010	00 0000	1111 0000	F0
1	00 0001	00 0001	1111 0001	F1
2	00 0010	00 0010	1111 0010	F2
3	00 0011	00 0011	1111 0011	F3
4	00 0100	00 0100	1111 0100	F4
5	00 0101	00 0101	1111 0101	F5
6	00 0110	00 0110	1111 0110	F6
7	00 0111	00 0111	1111 0111	F7
8	00 1000	00 1000	1111 1000	F8
9	00 1001	00 1001	1111 1001	F9
?	00 0000	00 1100	0110 1111	ALL OTHER CODES (see notes)

NOTES

a. EBCDIC 0100 1110 also translates to BCL 11 1010.

b. EBCDIC 1111 is translated to BCL 00 0000 with an additional flag bit on the most significant bit line (8th bit). This function is used by the unbuffered printer to stop scanning.

c. EBCDIC 1110 0000 is translated to BCL 00 0000 with an additional flag bit on the next to most significant bit line (7th bit). As the print drums have 64 graphics and space this signal can be used to print the 64th graphic. The 64th graphic is a "CR" for BCL drums and a "¢" for EBCDIC drums.

d. The remaining 189 EBCDIC codes are translated to BCL 00 0000 (? code).

e. The EBCDIC graphics and BCL graphics are the same except as follows:

	BCL	EBCDIC
1)	≥	' (single quote)
2)	x (multiply)	!
3)	≤	¬ (not)
4)	≠	_ (underscore)
5)	~	|

[1] Reproduced from ["Burroughs B6500 Information Processing Systems Reference Manual No. 1043676" September 1969, Appendix E]

states, the mapping is not 1 to 1. Therefore data may be lost. The 7-track tape generated is in odd parity.

b) TRTCH = ET - This is the same as TRTCH = T except that the tape generated is even parity.

c) TRTCH = E - The first two high order bits of the EBCDIC code are lost. Only the low order 6 bits are kept. In some computer installations this is the default mode. This means lost data. The tape generated is even parity

d) TRTCH = C - Data conversion is used. An EBCDIC character is written on a seven track tape without loss of data. This is the most acceptable form as it allows the recipient the capability of mapping the source graphic code to his target graphic code as he wishes. The tape is written in odd parity.

Since there are four options, only one of which is acceptable unless additional information about the data is known, the user often receives a tape with missing data that cannot be reconstructed. If data is missing, the user must request a new tape.

Under the SCOPE system on the CDC 6000 series, tapes can be in an UPDATE format. This is a common way for Control Data to send out modifications to their software.

A computing center may also elect to have this as a standard way.of transfering data. If the tape recorded in update format had a program on it that a user wished to get running on another manufacturer's computer, he would have to understand just how to extract enough meaningful control information in order to reconstruct the original line images.

Another major source tape generation problem is sending incorrect information with the tape. The sender may incorrectly state the number of files, physical and logical records and characters per physical record. Parity is rarely mentioned and occasionally the density is not correct. This makes the task of transforming the tape more difficult.

Problems also occur when a 9-track tape arrives at an installation that has only 7-track tape drives. An intermediate computer is necessary to copy the 9-track tape to a 7-track tape. It is quite possible that during this copy procedure, data may be added or lost. Thus when the tape reaches the target computer, it may have two levels of confusion involved.

A SYNTHESIS OF TEST CASES

Twenty test cases have been chosen for the purpose of discussing and illustrating some of the existing problems of magnetic tape portability. They are numbered according to the computer manufacturer of the source computer and the order in which they arrived (see Table 3.11). The test cases were not solicited but were actual instances of users who were not able to use the data in its original form and thus needed the data transformed to a useable form. Of the twenty cases, 15 different sending groups were represented. There were 16 individual recipients of these source tapes representing 12 different groups. Seven different computer manufacturers were involved in producing the various source tapes.

In 17 of the test cases, some form of graphic transformation was required. Excluding case C1, the target graphic was CDC 6500 Display code. In case C1, the target graphic was EBCDIC. Two cases(D1,M1) had ASCII as the source graphic, 2 cases(U1,U2) had FIELDATA as the source graphic, 5 cases(I1,I4,I5,I8,S1) had versions of EBCDIC as the source graphic and 7 cases(H1,I2,I6,I7,I10,I11,U3) had versions of BCD as the source graphic. In all of these cases the exact graphic transformation was not known in advance but was determined iteratively.

One non-graphic case(I9) involved floating point transformations, case I6 required signed numeric transformations and case I3 would have used the packed decimal option if all of the data were correct. In addition, case D1 also had one file involving fixed point transformations.

In many cases the graphic code expected (i.e. a verbal request or a letter that came with the tape) was different from the actual graphic code found on the tape. In particular, case I5 was to have been 2 files each of BCD and EBCDIC but really had 4 files of EBCDIC. Case I11 was to have been EBCDIC and it turned out to be BCD. In addition, graphic symbols that were not expected were found in all cases requiring graphic transformation.

One case(U3) involved two transformations of data. The tape was BCD which had been trasformed from FIELDATA. The binary counters that contained the length in words of the card image were thus treated as FIELDATA characters and transformed to BCD which altered the numeric value.

Logical end of files were found in cases U1,U2,U3. Case I2, produced by the Cambridge Monitor System, had binary card images as well as source card images mixed on one physical file. Several cases had to have their data decompressed(D1,M1,U1,U2,U3). Only case D1 involved the interchanging of bit patterns while deleting unnecessary bits was required by cases D1, I2, I8, I9, M1, U1, U2, and U3.

Logical records that spanned across physical records were involved in cases I6, I7, I8, U1, U2, and U3. Due to the structure of the source tape and constraints of the target operating system, case I7 required that logical records be mapped to a different logical arrangement on the target system. Trailer labels in each physical record had to be removed in cases U1, U2 and U3.

During conversion from 9-track to 7-track case I9 had the first physical record containing data removed. Case I8 had a physical record that was damaged. Fortunately it was possible to reconstruct this record by refering to a source listing. Two out of 8 bits were missing in case I3 as well as having a blocking factor different from that requested.

In conclusion, there were no cases in which it was possible to transform the tape based on the original information supplied by the user.

Additional details for the test cases may be found in Tables 5.6 and 5.7 in Chapter 5.

Table 3.11 Test Cases

CASE	SOURCE COMPUTER	GROUP SENDING	GROUP RECIEVING
I1	IBM 360	Argonne Labs	Nuclear Engineering
C1	CDC 6500	Purdue University Computing Center	Dusquesne Light
I2	IBM 360	MIT	Civil Engineering
M1	Modcomp III	Modular Computer Systems	Computing Center
I3	IBM 370/155	Administrative Data Processing	Freshman Engineering
S1	Siemens	University of Freiburg	Computer Sciences
U1	Univac 1108	University of Wisconsin	Computer Sciences
U2	Univac 1108	Texas Water Authority	Industrial Administration
I4	IBM 360/67	University of Michigan	Computer Sciences
I5	IBM 360	Argonne Labs	Nuclear Engineering
I6	IBM 7074	Bureau of Labor Statistics	Economics
M2	Modcomp III	Modular Computer Systems	Computing Center
I7	IBM	Investors Management Services	Industrial Administration
I8	IBM 370/165	McDonnell Douglas	Mechanical Engineering
I9	IBM 370/165	Bell Labs	Electrical Engineering

Table 3.11, cont.

U9	Univac 1106	University of Utah	Industrial Administration
I10	IBM 370	Bell Labs	Sociology
I11	IBM 360/67	University of Michigan	Computer Sciences
H1	Honeywell 6000	Bell Labs	Sociology
D1	PDP 11/45	Biology	Biology

CHAPTER 4.

AN APPROACH FOR PORTABLE MAGNETIC TAPES

GENERAL COMMENTS

This chapter considers the requirements that must exist to provide transformation of data from a source tape to a target tape. Six data types are discussed followed by their table specifications. Other requirements for data description are then discussed followed by a general algorithm for magnetic tape data transformation.

DATA TYPES

There are six required data types. They are: (1) null, (2) straight binary, (3) fixed point, (4) floating point, (5) packed, zoned, numeric decimal codes and (6) graphic codes.

Null

A Null type used to specify a null input field results in not transforming the data specified in the field. It is quite useful in editing out information. Used in conjunction with the concatenation feature, which is discussed later, it allows the joining of two non-adjacent fields. The option of specifying a null output field is necessary to allow the use of binary counters without passing them on as target fields.

Straight Binary

Straight Binary refers to normal binary fields consisting of zeros and ones. They are assumed to be converted directly. The ones stay ones and the zeros stay zeros. The main use is for passing identical bit formations between computers. This allows manipulation of the data to be done at a later time by the user within the target computer system.

Fixed Point

Transformation between single precision source fixed point and target fixed point is a minimum requirement. The fixed point attributes are contained in the computer attribute table that is briefly discussed later in this chapter. Attributes such as length of the integer and sign position can be used to isolate the functional parts of the fixed point number. A general algorithm can be constructed for fixed point regardless of the negative representation of numbers (i.e. ones-complement, twos-complement or sign-magnitude). Step 14, discussed later in the chapter, outlines this algorithm.

Floating Point

Transformations between source floating point and target floating point take place via an algorithm using the floating point attributes that are contained in the computer attribute table that is briefly described later in this chapter. Attributes such as length of the characteristic and mantissa, sign position of the characteristic and mantissa, base of the mantissa and characteristic, and method of negative representation can be used to isolate the functional parts of the floating point number. The minimum transformation requirements are to handle single precision normalized numbers.

Zoned, Packed and Numeric Decimal Codes

This code transformation type provides the capability to convert source packed, zoned or numeric decimal data to formats acceptable to the target computer. The packed decimal format can be considered to be a string of n bit digits with the rightmost digit being the sign. The normal transformation would be to a leading sign followed by a string of digits. The length of each target digit is controlled by the target computer.

The zoned and numeric decimal formats can be considered to be a string of m bit digits with the rightmost digit containing the sign as well as the value of the digit. The normal transformation would be as in packed decimal.

Graphic Codes

This type allows a numeric code representing a source graphic to be transformed to a numeric code representing the same graphic symbol on the target computer. Chapter 3 discussed this problem in depth.

TABLE REQUIREMENTS

The tape transformation algorithm described later in this chapter requires five tables from which the necessary attributes can be extracted for input and output type verification, computer descriptions and code transformations. Examples of implemented tables can be found in Chapter 5.

Master Graphic Table

This table consists of a basic set of graphic symbols and their corresponding codes for code transformation. It provides a mapping from one character set to another. The names for the graphic symbols should be based on the ANSI standards (ANSI 1967). Additional names can be added without benefit of standardization. The table should also be constructed so that it may be easily modified and contain, as a minimum, several well used codes such as ANSI, EBCDIC, BCDIC, ISO, DISPLAY and FIELDATA (see Tables 3.1, 3.3, 3.4, 3.5, 3.7, 3.8).

Packed and Zoned Decimal Table

This table is similar to the master graphic table except it allows code transformations for packed and zoned decimal to other graphic codes.

Computer Attribute Table

This table contains the attributes of a computer that are necessary to provide correct transformations for all data types. It must contain, as a minimum, (1) attributes to properly define fixed and floating point numbers including their negative number representation and (2) number of bits per word, character, byte and digit.

Type Transformation Table

This table relates a type name to a type number and the number of bits per graphic symbol.

Allowable Type Transformation Matrix

This is an input-output matrix that exists to allow transformations between different types.

OTHER REQUIREMENTS FOR DATA DESCRIPTION

For proper data description, other options that allow concatenation, repetition, flow control, sentinel checking, output, n bit to m bit transformation, labels and spanning are required. A field descriptor containing some of the above options can be used to define a field or number of fields.

Concatenation

Concatenation allows two fields to be combined to form a third. It is useful in transforming fields whose target description is different than that of the source description. Another use is in changing zoned, packed and numeric decimal fields so that they are acceptable for normal Fortran format statements.

Repetition

Repetition allows the value found in a field to be saved and used as a constant which will later control the number of times a field descriptor is to be repeated. It allows transforming variable length records or fields where the length of the record or field is embedded in a previous record or field.

Flow Control

Flow Control allows the user to control the flow of his input and output data description by specifying the field that will be transformed after the current field descriptor is satisfied. It reduces the number of field descriptors for a given run and can be used to repeat patterns by jumping over field descriptors and returning to them later.

Sentinel Checking

Sentinel Checking allows the user to specify a field as a control or sentinel field. The value specified in the sentinel value portion of the field descriptor is tested each time the field occurs. If the sentinel value is satisfied, the action specified in the sentinel action portion of the field descriptor is taken. Sentinels allow lines, records and files to be variable length.

Output

The output option allows a record to be written on the target tape upon completion of the transformations specified in the field descriptor. If the output option is not used, the transformed data specified by the field descriptor will be placed in the output buffer until another field descriptor requests output. Also, if the output option is not used in any field descriptor, the default will be to generate one physical record out for each physical record in.

N Bit to M Bit Transformation

This option allows the user to map n bit fields to m bit fields. Character sets do not always contain the same number of bits per character. Eight bit EBCDIC often has to be transformed to a 6-bit code such as BCD or DISPLAY and 6-bit codes must often be transformed to 8-bit codes. Word sizes also differ between computers.

Header and Trailer Labels

Any grouping of data may have header and/or trailer labels. Header labels may consist of information that allows identification of the following data. Trailer labels may contain check sum information. Tapes can have header and trailer files. Files can have header and trailer records. Records can have header and trailer fields. The capability must exist to either extract the necessary information from the header or trailer or to bypass them.

Spanning Across Physical Records

Spanning occurs when a logical record or field is contained in more than one physical record. Therefore the capability must exist to buffer the input in such a way that spanning is allowed since source data descriptions will differ from target data descriptions.

AN ALGORITHM FOR A TAPE TRANSFORMATION PROGRAM

The algorithm that follows is a procedure for transforming a tape generated on a source computer to one acceptable for a target computer. It is designed to reside on the target computer. The algorithm is composed of steps which are in turn composed of tasks. The notation for step i task j is step i.j. Normal flow through this algorithm is sequential unless otherwise noted. A global flow diagram of the relationship between steps may be found in Figure 4.1.

Since data description errors compound themselves which in turn results in erroneous output later, error diagnostics often terminate a job and return control to the user. Experience has shown that an incremental advancement through the data description is presently the best technique.

The algorithm is limited in that the fixed point step has not been used in actual production and the floating point step, although general in nature, has worked only for IBM 360 single precision floating point.

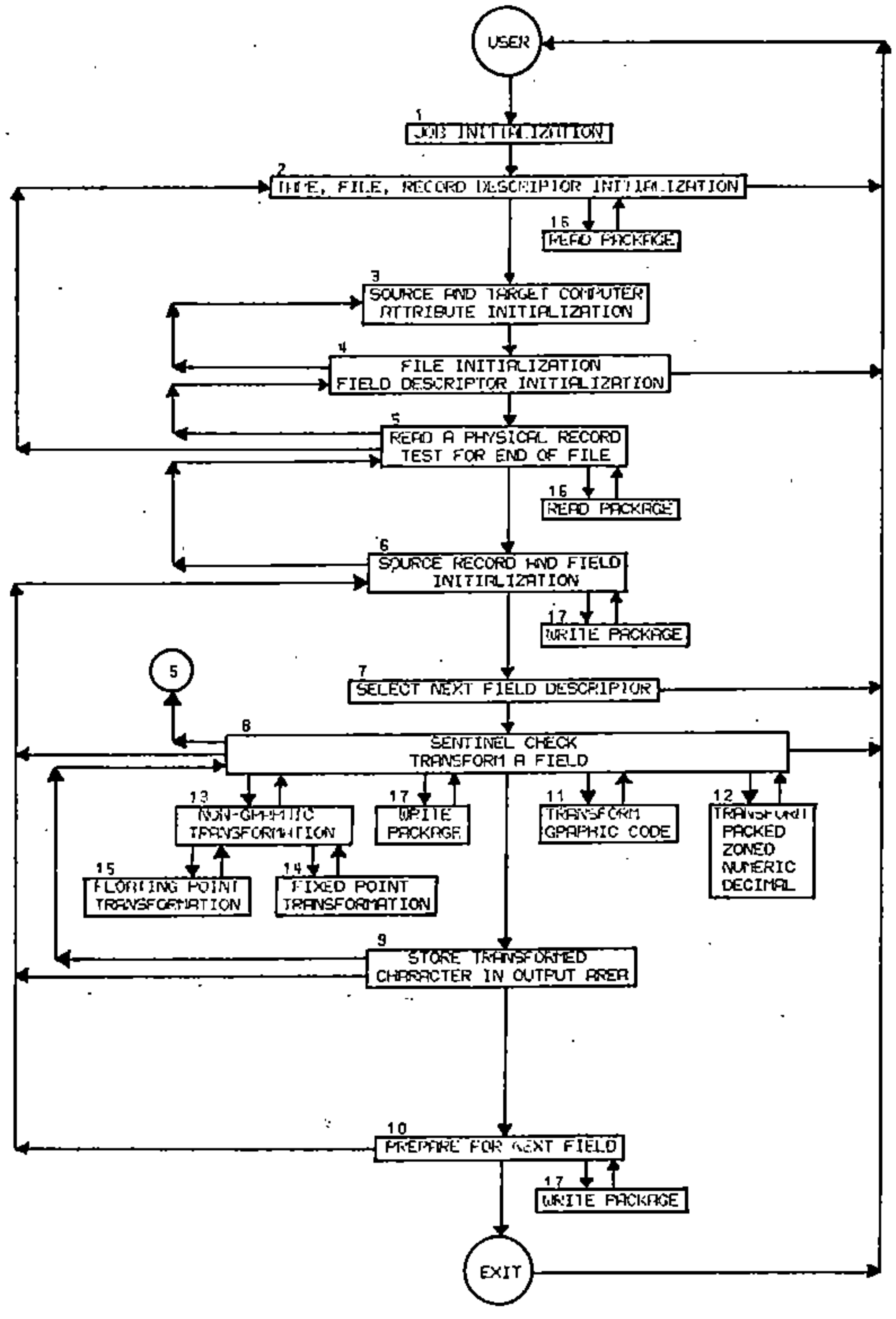

USER
1 JOB INITIALIZATION
2 TIME, FILE, RECORD DESCRIPTOR INITIALIZATION
16 READ PACKAGE
3 SOURCE AND TARGET COMPUTER ATTRIBUTE INITIALIZATION
4 FILE INITIALIZATION FIELD DESCRIPTOR INITIALIZATION
5 READ A PHYSICAL RECORD TEST FOR END OF FILE
16 READ PACKAGE
6 SOURCE RECORD AND FIELD INITIALIZATION
17 WRITE PACKAGE
5
7 SELECT NEXT FIELD DESCRIPTOR
8 SENTINEL CHECK TRANSFORM A FIELD
13 NON-GRAPHIC TRANSFORMATION
17 WRITE PACKAGE
11 TRANSFORM GRAPHIC CODE
12 TRANSFORM PACKED ZONED NUMERIC DECIMAL
15 FLOATING POINT TRANSFORMATION
14 FIXED POINT TRANSFORMATION
9 STORE TRANSFORMED CHARACTER IN OUTPUT AREA
10 PREPARE FOR NEXT FIELD
17 WRITE PACKAGE
EXIT

Figure 4.1 Global Diagram for a Tape Transformation Algorithm

Step 1 - Job Initialization

Task 1 - Read in Tables

Set up the input and output tapes, print program titles, read in type transformation options and print them out, read in and print out the allowable type transformation matrix, read in and print out the master graphic table, and the zoned and packed decimal table along with their headers.

Task 2 - Read in Header and Job Descriptors

Read in and print out general information (user name, address, project, etc.), the source computer and target computer.

Step 2 - Tape, File, Record Descriptor Initialization

Task 1 - Read Source and Target Tape Descriptions

Read in the number of tracks, density and number of files. If the number of tracks is zero, terminate the job by going to step 10.5, otherwise set up the density for the read physical record package by going to step 16.1.

Task 2 - Read Source File Description

Read in the parity, number of physical records per file, number of bits per character on the source and target computer, and the number of bits per digit on the source computer.

Task 3 - Read Source and Target Record Description

Read in and print out the blocking factor (logical records per physical record) and the number of leading bits to remove from each physical record.

<u>Step 3 - Source and Target Computer Attribute Initialization</u>

Load the source and target attribute tables from the computer attribute table. If the values of source bits per character, target bits per character, and source bits per digit are omitted or zero, take the corresponding values from the computer attribute table. If they have values, over-ride the corresponding computer attribute. Generate a table of masks to be used for bit manipulation. Set the file counter to one.

<u>Step 4 - File Initialization and Field Descriptor Initialization</u>

Task 1 - Header Deletion Initialization

If header bits are to be removed from each physical record, set initial input bit pointer and initial input word counter to exclude these leading bits.

Task 2 - Source File Initialization

set field descriptor area, input area and output area to zero, the output word counter to zero, input and output physical record counters to zero, field descriptor output switch to off.

Task 3 - Read and Verify Source and Target Field Description

Read in the field descriptors containing, the starting field, ending field, number of units, units, input type, output type, repetition option, concatenation option, output option and flow control option. Verify that the input and output types are transformable. If they are not, print out a message and terminate the job by going to step 10.5. If any of the types in the field descriptors are of the type fixed point, check to see if the word length of the source computer is greater than than that on the target computer. If it is, print out an appropriate message and terminate the job by going to step 10.5.

Task 4 - Target Field Initialization

If the output option in any field descriptor is on, set all output control to be triggered by the field descriptors.

<u>Step 5 - Read a Physical Record and Test for End Of File</u>

Task 1 - Check For Exceeding Physical Record Limit

If the number of physical records per file is unknown, go to step 5.2. If not, check to see if the number of physical records for this file are to be exceeded. If they are, go to step 5.4, otherwise continue.

Task 2 - Read One Physical Record

Increase the input physical record counter by one. Read one physical record by going to step 16.2.

Task 3 - Test For End Of File

Check for an end of file. If an end of file has not occurred on the source tape, go to step 6.1, otherwise continue.

Task 4 - File Termination Procedures

Write an end of file on the target tape. If graphic transformations have been made, print out the graphics that were not transformable and the number of times that each of them has occurred. Increase the file counter by one. If the file counter is greater than the total number of files to be transformed go to step 2.1, otherwise go to step 4.2.

Step 6 - Source Record and Field Initialization

Task 1 - New Record Initialization

Turn the initialization switch on. Set the input word counter equal to the initial input word counter, the logical record counter to zero, and the output word counter to zero.

Task 2 - Field Initialization

Set the field descriptor counter to zero, the repetition counter to zero and the field counter to zero.

Task 3 - Advance to Next Field and Test For Logical Record End

Increase the field counter by one. Save the current source bits per unit. If this is not the end of a logical record go to step 7.1. If this is a repeating field that is not completed, go to step 7.4. Increase the logical record counter by one. If the number of logical records per physical record is not equal to the logical record counter go to step 6.2.

Task 4 - Output Option Test

If the output is controlled by field descriptors, go to step 5. Otherwise, write out one record by going to step 17.

Task 5 - Reset Output Counter

Set the output word counter to zero. Return to step 5.

<u>Step 7 - Select Next Field Descriptor</u>

Task 1 - Set Up Source Bits Per Unit

If all fields described by the field descriptor have not been transformed, go to step 7.4, otherwise select the next field descriptor. Set up the source bits per unit based on the requested units (bits, characters, bytes, digits, words). If the units are unknown, write out an appropriate message and terminate the job by going to step 10.5.

Task 2 - Multiple Unit Test

If the input type is not straight binary, fixed point or floating point, go to step 7.4, otherwise continue.

Task 3 - Set Up Total Source Bits Per Unit For This Field

Set source bits per unit equal to the product of source bits per unit and the number of units in the current field descriptor. Set the number of units to be transformed equal to one. Go to step 8.1.

Task 4 - Set Up Number Of Units

Set the number of units to be transformed equal to the number of units in the field descriptor.

<u>Step 8 - Sentinel Check, Transform a Field</u>

Task 1 - Transformation Initialization

Initialize the character pointer to one. If the initialization switch is off, go to step 9.5, otherwise set the initialization switch to off.

Increase the output word counter by one. Clear the next word to be used for output, set the output bit pointer to the number of bits per word in the target computer and set the input bit pointer to the initial bit pointer.

Task 2 - Load Working Area With Input

Load the working area with the next word residing in the target computer to be transformed.

Task 3 - Extract The Bits To Be Transformed

Extract the bits to be transformed.

Task 4 - Sentinel Field Check

If this field is not a sentinel go to step 8.6. If the value in the field is not the required sentinel go to step 8.6. Otherwise, write out one record by going to step 17.

Task 5 - Check Type Of Sentinel

Set the output word counter to one. If this is an end of file sentinel go to step 5.4. Otherwise, treat as end of logical record sentinel and go to step 6.2.

Task 6 - Choose Correct Transformation

If the input type is graphic, go to step 11. If the input type is packed, zoned or numeric decimal, go to step 12. If the input type is null, go to step 9.2. Otherwise go to step 13.

Task 7 - Store Transformed Field

Load the current output word with the transformed field. If this value is not to be saved for use as a repetition constant for future fields go to step 10, otherwise save the product of this value and the multiplier value in the repetition constant field of the field descriptor. Go to step 10.

<u>Step 9 - Store Transformed Character In Output Area</u>

Task 1 - Output Area Initialization

Mask out all but the necessary bits of the transformed character. If the output does not go across word boundaries on the target computer, go to step 9.3. Otherwise increase the output bit pointer by the number of bits per word of the target computer less the number of target bits per unit. Put the leftmost bits that will fill up the current output word into the current output word. Increase the output word counter by one. Place the remaining output bits in this new output word.

Task 2 - Increase Output Bit Pointer

Increase the output bit pointer by the number of target bits per unit. Go to step 9.4.

Task 3 - Place Transformed Character In Output Area

Shift the new output character left by the difference between the output bit pointer and the target bits per unit and place it in the output word.

Task 4 - End Of Field Test

If the current character counter is greater than or equal to the number of units to be transformed, go to step 10. Otherwise, increase the character pointer by one and save the current source bits per unit.

Task 5 - Advance To A New Field

Reduce the input bit pointer by the saved current source bits per unit, reduce the output bit pointer by the number of target bits per unit. If the output bit pointer is not equal to zero, go to step 9.6, otherwise continue. Set the output bit pointer equal to the target bits per word. Increase the output word counter by one. Zero out the new output word.

Task 6 - Advance Input And Check For End Of Physical Record

If the input bit pointer is greater than or equal to the number of source bits per unit, go to step 8.3. Otherwise, increase the input word counter by one. If the input word counter is less than the number of words read in by the read package (step 16) go to step 9.7, otherwise print out a message that fewer logical records were found than specified. Go to step 6.4.

Task 7 - Process Input Across Word Boundaries

If the input bit pointer is equal to zero, set the input bit pointer to the number of bits per word in the target computer and go to step 8.2. Otherwise, save the remaining bits that need to be transformed, bring the next input word into the working area, and extract the necessary bits that need to be added to the previous bits. Increase the input bit pointer by the number of bits per word in the target computer. Go to step 8.4.

Step 10 - Prepare For Next Field

Task 1 - Field Descriptor Repeat Test

If the current field output type is null, go to step 10.3. If this field descriptor is not to be repeated, set the repetition counter to zero and go to step 10.2. Otherwise, increase the repetition counter by one, if the field has been repeated enough times, reset the repetition counter to zero and go to step 10.2, otherwise continue. Reduce the field descriptor counter by one, reduce the field counter by one. If the concatenate switch is on, go to step 6.3. Otherwise go to step 10.2.

Task 2 - Field Concatenation Test

If the concatenate switch is on go to step 6.3. Otherwise, increase the output word counter by one, zero out the new output word.

Task 3 - Output Option Test

If the concatenate switch is on, go to step 6.3. Otherwise, set the output bit pointer to the sum of the bits per word of the target computer and the number of bits per unit on the target computer. If the field descriptor output option is off, go to step 6.3. Otherwise, output by going to step 17.

Task 4 - Initialize Output Word Counter

Set the output word counter to one. Go to step 6.3.

Task 5 - Terminate The Job

Write two end of file marks on the output tape, print a message that the run is terminated. Terminate this job by returning control to the user.

<u>Step 11 - Transform One Graphic Code</u>

Task 1 - Transform One Graphic Code

Look up the target graphic corresponding to the source graphic symbol in the master graphic table. If it is undefined, print out a message to this effect and increase the corresponding error count for this code by one, otherwise continue. Go to step 8.7.

<u>Step 12 - Transform Packed, Zoned or Numeric Decimal</u>

Task 1 - First Character or Digit

If this is not the first character or digit of the field go to step 12.2. Otherwise, save the output word and bit pointers for the future placement of the sign, return zeros for the transformed value, go to step 12.6.

Task 2 - Normal Transformation

If this is the last character or digit of the field go to step 12.3. Otherwise, return the last character or digit transformed, go to step 12.6.

Task 3 - Last Digit is Packed Decimal Sign

If this is not a packed decimal transformation, go to step 12.4. Otherwise, return the last digit transformed, look up the corresponding value for the sign of the target computer in the zoned-packed decimal table, place the sign in the position saved in step 12.1, go to step 8.7.

Task 4 - Transform Sign for Zoned or Numeric Decimal

Return the last character transformed. If the sign has already been transformed go to step 12.5, otherwise save the input word and bit pointers, reduce the character counter by one, separate the sign from the character and save the character for step 12.5. If input type is zoned decimal look up the corresponding character for the sign of the target computer in the zoned-packed decimal table, otherwise look the sign up in the master graphic table. Place the sign in the position saved in step 12.1, go to 12.6.

Task 5 - Transform Last Character for Zoned Decimal

Restore the input word and bit pointers to the correct position, use the character in step 12.4 and go to step 12.6.

Task 6 - Look Up Transformed Character or Digit

Look up the corresponding digit or character for the target computer in the packed and zoned decimal table, save this transformed value for the next time, go to step 8.7.

<u>Step 13 - Non-Graphic Transformation</u>

Task 1 - Fixed Point, Floating Point or Straight Binary Test

If the input type is straight binary, set the target field equal to the source field and go to step 13.2. If the input type is fixed point go to step 14. If the input type is floating point go to step 15. Otherwise print out an error message and continue.

Task 2 - Return

Go to step 8.7.

Step 14 - Fixed Point Transformation

Task 1 - Test For Negation

Save the sign bit. If it is a 0 (positive), set the target field equal to the source field and go to step 14.5, otherwise extract the integer part.

Task 2 - Source Computer Is Sign-Magnitude

If the source computer is sign-magnitude, set the target field equal to -(integer part) and go to step 14.5, otherwise continue.

Task 3 - Source Computer Is Ones-Complement

If the source computer is ones-complement and the target computer is sign-magnitude, set the target field equal to -(integer part + 1) and go to step 14.5, otherwise continue. If the source computer is ones-complement and the target computer is also ones-complement, set the target field equal to -(integer part), set the sign bit to a 1 and go to step 14.5, otherwise continue.

If the source computer is ones-complement and the target computer is twos-complement set the target field equal to (integer part - 1), set the sign bit to a 1 and go to step 14.5, otherwise continue.

Task 4 - Source Computer Is Twos-Complement

If the source computer is twos-complement and the target computer is ones- complement, set the target field equal to (integer part + 1), set the sign bit of the output field equal to 1 and go to step 14.5, otherwise continue. If the source computer is twos-complement and the target computer is sign magnitude, set the target field equal to -(integer part), go to step 14.5, otherwise continue. If the source computer and the target computer are both twos-complement, set the target field equal to the integer portion with its sign bit a 1 and go to step 14.5, otherwise continue.

Task 5 - Return From Fixed Point

Return to step 13.2.

<u>Step 15 - Floating Point Transformation</u>

Task 1 - Floating Point Transformation

Extract the sign, exponent and fraction and convert them to target floating point. An algorithm for IBM 360 to CDC 6500 floating point may be found in Chapter 2.

<u>Step 16 - Read Package</u>

Task 1 - Set Up Density

If the density is allowable on the target computer, set up the density for step 16.2 and return to step 2.2, otherwise send an error message and terminate the job by going to step 10.5.

Task 2 - Read One Physical Record

Read one physical record in either odd or even parity and at any of the allowable densities, set the end of file switch on if a physical end of file appears, reread the physical record a sufficient number of times to verify its correctness. If it is correct, go to step 5.3, otherwise print out an error message, set the end of file switch on and go to step 5.3.

<u>Step 17 - Write Package</u>

Task 1 - Output One Record

Reduce the number of words to be written out by one, increase the output record counter by one, write out the record and file number on the print file, write out one record on the target tape in a format acceptable for the target computer.

Zero out the output area. If we came from step 6.4, go to step 6.5. If we came from step 8.4, go to step 8.5. If we came from step 10.3 go to step 10.4.

CHAPTER 5

IMPLEMENTATION OF THE APPROACH ON A CDC 6500

GENERAL COMMENTS

To verify the approach discussed in Chapter 4, a program was implemented on the CDC 6500. Fortran was chosen as the implementation language primarily due to its widespread availability. In addition, the Fortrans running under the MACE operating system at Purdue University are well maintained and reasonably efficient. Also, under the Minnesota Fortran Compiler (MNF) there is an option to allow checking for ANSI Fortran. Functions are also available that allow the necessary bit manipulation capabilities.

The majority of the approach as discussed in Chapter 4 has been implemented. Only those features that actually arose in production test cases were included. It was felt that adding features for hypothetical cases was not valid, particularly when it was a major effort to include the features for actual test cases.

The program has been compiled and executed under the RUN, FUN and MNF compilers. The program compiles under standard ANSI Fortran with the two following exceptions:

1. The use of ENTRY in subroutines.
2. The use of O(octal) in Format statements.

However, the use of ENTRY statements as used in the program is generally available in most Fortrans. The use of octal can be changed depending on whether the graphic tables and their errors are to be input and output in octal, decimal or hexadecimal.

The program consists of the main program, 14 subroutines and 6 functions. Each of these are discussed with implementation assumptions and limitations. Figure 5.1 illustrates a flow network of the tape transformation program.

MAIN PROGRAM

The main program extracts the bits that are to be transformed and acts as a driver for the subroutines. It is machine independent excluding dimension and common statements which must be changed when ten 6-bit characters do not fit into one word.

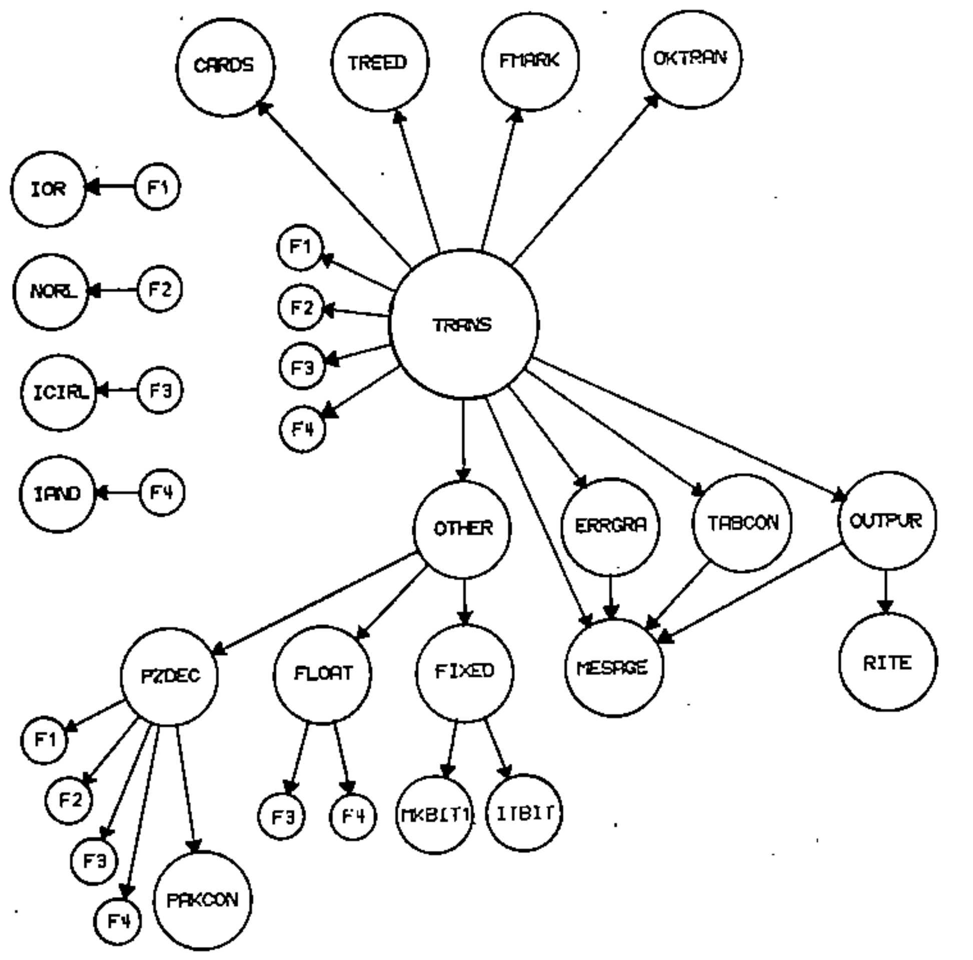


Figure 5.1 Flow Network of the Tape Transformation Program

MACHINE DEPENDENT SUBROUTINES

Six subroutines (CARDS, FLOAT, MESAGE, RITE, TABCON, TREED) are machine dependent. They use system routines that are available under MACE, rely on a 60-bit word or take advantage of the 63 character set on the CDC 6500.

CARDS

Subroutine CARDS has ten entry points and handles the reading of all data cards. It prints out the input and provides some error diagnostics. The range of the values in the data descriptor cards have been chosen to allow a specific number of variables to exist on each card. If necessary, the formats of these cards can be changed to accomodate larger values. However, in the case of alphanumeric fields, the dimension of the variable should be examined and made larger if required.

FLOAT

Subroutine FLOAT handles IBM 360/370 32-bit floating point transformations. It is called from subroutine OTHER. The fractional part is extracted and the binary point moved so that it can be treated as an integer. The sign and exponent are also extracted. the resulting number is then the product of the integer and the exponent modified by the sign. The algorithm was discussed in Chapter 2. The implementation is based on a 60-bit word and therefore is machine dependent.

MESAGE

Subroutine MESAGE handles the majority of the messages that are placed in the print file. The only other subroutines that print out messages are CARDS and TREED. It is called from the main program as well as various subroutines. Because of the nature of ten character words and octal number capability on the CDC 6500, this routine is machine dependent.

RITE

Subroutine RITE writes on to the target tape. The format is presently restricted to a formatted write. This is machine dependent and must be modified for a particular system convention. It is called from subroutine OUTPUR.

TABCON

When subroutine TABCON is given the source graphic code, the corresponding target graphic code is returned. Table look-up is performed on the master graphic table by means of a pointer, since the table is already pre-sorted on the input type. Table 5.3 contains the available graphic symbols and their corresponding numeric codes. All symbols in the master graphic table that do not have codes are preset to all octal sevens, which is a minus zero on the CDC 6500. If the resulting code of the target graphic is not less than plus zero or greater than 255 it is assumed to be correct. Otherwise an error message is triggered prior to the output of the field. It is called from the main program.

TREED

This subroutine has two entry points. PREED sets up the density to be used in TREED. If the density requested is other than 200, 556 or 800 bits per inch the run is terminated.

TREED reads one physical record from magnetic tape. If an end-of-file occurs, a flag is set which will be handled by the main program. The number of 60-bit words read in is also returned. It is capable of reading either even or odd parity tapes with densities of 200, 556, or 800 bits per inch. No attempt is made to read mixed parity files or mixed density tapes. If a tape error occurs, control is returned to the main program.

This subroutine is machine dependent since the call to the read one physical record (READ) routine available under the MACE operating system has machine dependent arguments. READ also detects an end-of-file and abnormal conditions. TREED will have to be modified to conform with the interface for each computer installation. Both PREED and TREED are called from the main program.

MACHINE INDEPENDENT SUBROUTINES

Eight subroutines (ERRGRA, FIXED, FMARK, OKTRAN, OTHER, OUTPUR, PAKCON, PZDEC) are machine independent.

ERRGRA

Subroutine ERRGRA writes out, after an end of file has occurred, all of the graphics and their numeric codes that were used as a source graphic but were undefined as a target graphic. For example, the graphic "ampersand" is not defined on some character sets. It is called from the main program.

FIXED

Subroutine FIXED is called from subroutine OTHER. It handles all fixed point transformations. If the number is negative, that is a one bit in the sign position, the computer attribute tables for both the source computer and the target computer are used to establish if sign-magnitude, ones complement or twos complement arithmetic is used and then the appropriate action is taken. The algorithm was discussed in Chapter 4. The integer length of the source computer must be no larger than the integer length on the target computer. It is assumed that the sign bit is the leftmost bit in the field to be transformed. This subroutine has never been used in actual production.

FMARK

Subroutine FMARK writes an end of file on the target tape. It is called from the main program.

OKTRAN

Subroutine OKTRAN checks to see if the input data type and the output data type are transformable. Types greater than or equal to ten are assumed to be graphic and are always transformable to one another. If either type is greater than 25 an error occurs. Types are checked against the allowable type transformation matrix (see Table 5.2) for acceptability. If a 1 is returned it is acceptable. If a 0 is returned it is not acceptable. It is called from the main program.

OTHER

Subroutine OTHER is called from the main program and acts as a driver for all types other than null, graphic, packed, zoned and numeric decimal. It handles straight binary, fixed point, and floating point types. In the case of fixed point, it issues a call to FIXED. In the case of floating point, it issues a call to FLOAT. If other transformation types occur, an error message is printed out.

OUTPUR

Subroutine OUTPUR is called from the main program. It updates the output record counter, writes out the target record number and file number on the print file and provides the correct number of words to be written out. It issues a call to subroutine RITE which writes out 1 record.

PAKCON

Subroutine PAKCON is called by surroutine PZDEC. Given the source graphic code, the corresponding target graphic code is returned by table look-up (see Table 5.4). It operates similarly to TABCON without the error messages.

PZDEC

Subroutine PZDEC is called from the main program. It handles packed (IBM 360), zoned (IBM 360) and numeric (IBM 7074) decimal transformations. In zoned and numeric decimal, the sign is an overpunch on the last character. In packed decimal, the sign is the rightmost digit. On output, the sign is placed preceeding the leftmost digit. One more character is necessary on output than input.

FUNCTIONS

The bit manipulation capabilities have been implemented as machine dependent functions. IOR and IAND take advantage of CDC 6500 Fortran. ICIRL, INORL, MKBIT and ITBIT use functions programmed in assembly language that are available in the LOGIC package under the MACE operating system.

IAND

The function IAND performs a logical AND between all of the bits of two words. It is implemented using .AND. which is available under CDC 6500 Fortran.

IOR

The function IOR performs a logical OR between all of the bits of two words. It is implemented using .OR. which is available on CDC 6500 Fortran.

ICIRL

The function ICIRL performs a circular left shift of all bits including the sign bit. Bits leaving the sign bit are reentered on the right. The arguments are the location of the word to be shifted and the number of bits to be shifted left circularly. It is implemented using the ISHFTLA function available in the LOGIC package.

INORL

The function INORL performs a left shift with zero fill on the right. The arguments are the location of the word to be shifted and the number of bits to be shifted left.

It is implemented using the ISHFTL function available in the LOGIC package.

MKBIT1

The function MKBIT1 sets a single bit to a one. The arguments are the location of the word in which the bit is to be set and the position of the bit to be set to a one. Bit positions decrease from left to right. That is they go from 59 down to 0. It is implemented using the TBIT function available in the LOGIC package.

ITBIT

The function ITBIT returns the value of a single bit in a word. The arguments are the location of the word in which the bit is to be tested and the position of the bit to be tested. Bit positions decrease from left to right. That is they go from 59 down to 0. It is implemented using the IBITON function available in the LOGIC package.

TABLES

There are five major tables that were used during implementation. They are (1) the type transformation table, (2) the allowable type transformation matrix, (3) the master graphic table, (4) the packed and zoned decimal table and (5) the computer attribute table. The computer attribute table is located in the main program. The four other tables are read in as data cards. The tables may be modified to change graphic codes, add new graphic symbols, limit transformations between data types or add new computer attributes.

Type Transformation Table

The type transformation table consists of the type number, the type name and the number of bits per graphic character if the type is a member of the graphic type. There are 25 possible types, 2 of which are undefined to allow for further expansion. See Table 5.1.

Table 5.1 Type Transformation Table

TYPE NUMBER	TYPE NAME	BITS PER GRAPHIC CHARACTER
1	NULL	
2	STRAIGHT BINARY	
3	FIXED POINT	
4	FLOATING POINT	
5		
6		
7	NUMERIC DECIMAL(IBM 7074)	
8	ZONED DECIMAL(IBM 360)	
9	PACKED DECIMAL(IBM 360)	
10	DISPLAY (CDC 6500)	6
11	INTERNAL BCD (CDC 6500)	6
12	EXTERNAL BCD (CDC 6500)	6
13	BCDIC (ALTERNATE)	6
14	BCDIC (PRINCIPLE)	6
15	BCD BASIC	6
16	BCD BASIC + 7	6
17	BCD (TO BE MODIFIED)	6
18	BCD (UNIVAC 1108)	6
19	EXTERNAL BCL (B6500)	6
20	INTERNAL BCD (H635)	6
21	FIELDATA (UNIVAC 1108)	6
22	ISO	7
23	ASCII (7-BIT)	7
24	ASCII (8-BIT)	8
25	EBCDIC	8

Allowable Type Transformation Matrix

The allowable type transformation matrix is implemented as an input-output matrix (see Table 5.2). A 1 at the intersection of input type and output type represents a valid transformation while a 0 represents an invalid transformation. This table is referenced by subroutine OKTRAN. Type numbers are the same as in Table 5.1 with type 10+ representing types 10 to 25. Types 5 and 6 are available for future expansion.

Table 5.2 Allowable Type Transformation Matrix

		OUTPUT TYPE NUMBER									
		1	2	3	4	5	6	7	8	9	10+
INPUT TYPE NUMBER	1	1	0	0	0	0	0	0	0	0	0
	2	1	1	0	0	0	0	0	0	0	0
	3	1	0	1	0	0	0	0	0	0	0
	4	0	0	0	1	0	0	0	0	0	0
	5	0	0	0	0	0	0	0	0	0	0
	6	0	0	0	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	0	0	0	1
	8	0	0	0	0	0	0	0	0	0	1
	9	0	0	0	0	0	0	0	0	0	1
	10+	0	0	0	0	0	0	0	0	0	1

Master Graphic Table

The master graphic table consists of 16 graphic character sets and allows up to 400 graphic symbols (see Table 5.3). It has been constructed from graphics used in the test cases discussed in Chapter 3 plus tables from other manufacturers. The initial symbol names were chosen to be those specified by ANSI (FIPS PUB 1 1968). Other symbol names were added later. In the table, the numeric codes are placed on the same card as the graphic symbol name. This allows new numeric codes or completely new tables to be added to the existing master table. Once the source graphic code has been selected, this table is pre-sorted on the source graphic to provide easier table look-up by means of a direct pointer. The type numbers are the same as in Table 5.1.

Packed and Zoned Decimal Table

The packed and zoned decimal table is in the same format as the master graphic table. It consists of 16 graphic character sets and allows 16 graphic symbols (see Table 5.4). The graphics are the numerics (0 to 9) and 6 others for the plus and minus sign. The table is pre-sorted to IBM 360 notation with decimal 12 as plus and decimal 13 as minus. The type numbers are the same as Table 5.1.

Table 5.3 Master Graphic Table

	TYPE NUMBER															
	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
A	01	21	61	61	61	61	61	61	61	61	21	06	101	101	241	301
B	02	22	62	62	62	62	62	62	62	62	22	07	102	102	242	302
C	03	23	63	63	63	63	63	63	63	63	23	10	103	103	243	303
D	04	24	64	64	64	64	64	64	64	64	24	11	104	104	244	304
E	05	25	65	65	65	65	65	65	65	65	25	12	105	105	245	305
F	06	26	66	66	66	66	66	66	66	66	26	13	106	106	246	306
G	07	27	67	67	67	67	67	67	67	67	27	14	107	107	247	307
H	10	30	70	70	70	70	70	70	70	70	30	15	110	110	250	310
I	11	31	71	71	71	71	71	71	71	71	31	16	111	111	251	311
J	12	41	41	41	41	41	41	41	41	41	41	17	112	112	252	321
K	13	42	42	42	42	42	42	42	42	42	42	20	113	113	253	322
L	14	43	43	43	43	43	43	43	43	43	43	21	114	114	254	323
M	15	44	44	44	44	44	44	44	44	44	44	22	115	115	255	324
N	16	45	45	45	45	45	45	45	45	45	45	23	116	116	256	325
O	17	46	46	46	46	46	46	46	46	46	46	24	117	117	257	326
P	20	47	47	47	47	47	47	47	47	47	47	25	120	120	260	327
Q	21	50	50	50	50	50	50	50	50	50	50	26	121	121	261	330
R	22	51	51	51	51	51	51	51	51	51	51	27	122	122	262	331
S	23	62	22	22	22	22	22	22	22	22	62	30	123	123	263	342
T	24	63	23	23	23	23	23	23	23	23	63	31	124	124	264	343
U	25	64	24	24	24	24	24	24	24	24	64	32	125	125	265	344
V	26	65	25	25	25	25	25	25	25	25	65	33	126	126	266	345
W	27	66	26	26	26	26	26	26	26	26	66	34	127	127	267	346
X	30	67	27	27	27	27	27	27	27	27	67	35	130	130	270	347
Y	31	70	30	30	30	30	30	30	30	30	70	36	131	131	271	350
Z	32	71	31	31	31	31	31	31	31	31	71	37	132	132	272	351
0	33	00	12	12	12	12	12	12	12	12	00	60	060	060	120	360
1	34	01	01	01	01	01	01	01	01	01	01	61	061	061	121	361
2	35	02	02	02	02	02	02	02	02	02	02	62	062	062	122	362
3	36	03	03	03	03	03	03	03	03	03	03	63	063	063	123	363
4	37	04	04	04	04	04	04	04	04	04	04	64	064	064	124	364
5	40	05	05	05	05	05	05	05	05	05	05	65	065	065	125	365
6	41	06	06	06	06	06	06	06	06	06	06	66	066	066	126	366
7	42	07	07	07	07	07	07	07	07	07	07	67	067	067	127	367
8	43	10	10	10	10	10	10	10	10	10	10	70	070	070	130	370
9	44	11	11	11	11	11	11	11	11	11	11	71	071	071	131	371

Table 5.3, cont.

	TYPE NUMBER															
	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
BLANK	55	60	20	20	20		20	20	20	20	20	05	040	040	100	100
COMMA	56	73	33	33	33		33	33	33	33	73	56	054	054	114	153
PERIOD	57	33	73	73	73		73	73	73	73	33	75	056	056	116	113
SEMICOLON	77	37	77	56	56			56	56	56	56	73		073	133	136
COLON	63	12		15	15			15	15	15	15	53		072	132	172
QUESTION MARK				72	72			72	72	00	17	54	077	077	137	157
EXCLAMATION PT.				52	52			52	52		77	55	041	041	101	132
APOSTROPHE				14				14	14		57	72		047	107	175
QUOTATION MARKS										37	76			042	102	177
AMPERSAND					60				00	60	32	46	046	046	106	120
PERCENT					34				35	34	74	52	045	045	105	154
NUMBER SIGN					13				77	13	13	03		043	103	173
COMMERCIAL AT					14				17	14	14	00		100	240	174
LOZENGE					74				37			76				
CENT SIGN																112
DOLLAR SIGN	53	53	53	53	53		53	53	53	53	53	47		044	104	133
ASTERISK	47	54	54	54	54		54	54	54	54	54	50	052	052	112	134
SLANT	50	61	21	21	21		21	21	21	21	61	74	057	057	117	141
PLUS	45	20	60	60				60	60	72	60	42	053	053	113	116
MINUS	46	40	40	40	40		40	40	40	40	52	41	055	055	115	140
EQUALS	54	13	13	13				13	13	35	75	44	075	075	135	176
NOT EQUAL	64	14	14							32						
.LT.	72	32	72	76	76			76	76	76	36	43	074	074	134	114
.GT.	73	57	57	16	16			16	16	16	16	45	076	076	136	156
.LE.	74	15	15							57						
.GE.	75	35	75							17						

Table 5.3, cont.

TYPE NUMBER

	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
OPENING PAREN.	51	74	34	34				34	34	75	35	51	050	050	110	115
CLOSING PAREN.	52	34	74	74				74	74	55	55	40	051	051	111	135
OPENING BRACKET	61	17	17	75	75			75	75	74	12	01		133	273	
CLOSING BRACKET	62	72	32	55	55			55	55	36	34	02		135	275	
OPENING BRACE														173	373	
CLOSING BRACE														175	375	
UP ARROW	70	55	55								40					
DOWN ARROW	71	56	56													
RIGHT ARROW	65	75	35													
LEFT ARROW										77	72					
AND	67	77	37													
OR	66	52	52													117
NOT	76	36	76													137
IDENTITY	60	76	36													
REVERSE SLANT				36	36			36	36		37	57		134	274	
VERTICAL LINE														174	374	
OVERLINE(TILDE)														176	376	
UNDERLINE													100	137	277	155
GRAVE ACCENT													137	140	340	
CIRCUMFLEX														136	276	
X MULTIPLICATION										52						
MODE CHANGE DELT				57	57			57	57			04				
WORD SEPARATOR				35	35			35								
RECORD MARK HHV				32	32			32								
TAPE MARK				17	17			17								
GROUP MARK HHHV				77	77			77								
SEGMENT MARK				37	37			37								

Table 5.3, cont.

	TYPE NUMBER															
	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
LOWER CASE A													141	141	341	201
LOWER CASE B													142	142	342	202
LOWER CASE C													143	143	343	203
LOWER CASE D													144	144	344	204
LOWER CASE E													145	145	345	205
LOWER CASE F													146	146	346	206
LOWER CASE G													147	147	347	207
LOWER CASE H													150	150	350	210
LOWER CASE I													151	151	351	211
LOWER CASE J													152	152	352	221
LOWER CASE K													153	153	353	222
LOWER CASE L													154	154	354	223
LOWER CASE M													155	155	355	224
LOWER CASE N													156	156	356	225
LOWER CASE O													157	157	357	226
LOWER CASE P													160	160	360	227
LOWER CASE Q													161	161	361	230
LOWER CASE R													162	162	362	231
LOWER CASE S													163	163	363	242
LOWER CASE T													164	164	364	243
LOWER CASE U													165	165	365	244
LOWER CASE V													166	166	366	245
LOWER CASE W													167	167	367	246
LOWER CASE X													170	170	370	247
LOWER CASE Y													171	171	371	250
LOWER CASE Z													172	172	372	251

Table 5.3, cont.

	TYPE NUMBER															
	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
NUL													000	000		
SOH													001	001		
STX													002	002		
ETX													003	003		
EOT													004	004		
ENQ													005	005		
ACK													006	006		
BEL													007	007		
BS													010	010		
HT													011	011		
LF														012		
VT													013	013		
FF													014	014		
CR														015		
SO													016	016		
SI													017	017		
DLE													020	020		
DC1													021	021		
DC2													022	022		
DC3													023	023		
DC4														024		
NAK													025	025		
SYN													026	026		
ETB													027	027		
CAN													030	030		
EM													031	031		
SUB														032		
ESC													033	033		
FS													034	034		
GS													035	035		
RS													036	036		
US													037	037		
DEL													177	177	377	

Table 5.4 Packed and Zoned Decimal Table

	TYPE NUMBER															
	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	33	00	12	12	12	12	12	12	12	12	00	60	060	060	120	360
1	34	01	01	01	01	01	01	01	01	01	01	61	061	061	121	361
2	35	02	02	02	02	02	02	02	02	02	02	62	062	062	122	362
3	36	03	03	03	03	03	03	03	03	03	03	63	063	063	123	363
4	37	04	04	04	04	04	04	04	04	04	04	64	064	064	124	364
5	40	05	05	05	05	05	05	05	05	05	05	65	065	065	125	365
6	41	06	06	06	06	06	06	06	06	06	06	66	066	066	126	366
7	42	07	07	07	07	07	07	07	07	07	07	67	067	067	127	367
8	43	10	10	10	10	10	10	10	10	10	10	70	070	070	130	370
9	44	11	11	11	11	11	11	11	11	11	11	71	071	071	131	371
UNDEFINED																
UNDEFINED																
PLUS	45	20	60	60				60	60	72	60	42	053	053	113	116
MINUS	46	40	40	40	40		40	40	40	40	52	41	055	055	115	140
UNDEFINED																
UNDEFINED																

Computer Attribute Table

The computer attribute table contains the attributes of a computer necessary to properly define data that originated on the source computer. In the implemented program, the attributes listed in Table 5.5 have been included. There is ample space, 50 entries of which 20 have values, to allow additional attributes that were discussed in Chapter 4 to be included at a later time.

Table 5.5 Computer Attribute Table

ATTRIBUTE	COMPUTER: CDC 6500	IBM 360-370	UNIVAC 1108	HIS 635
BITS PER WORD	60	32	36	36
BITS PER BYTE	12	8		
BITS PER CHARACTER	6	6	6	6
BITS PER DIGIT		4		
INTEGER				
SIGN BIT POSITION	1	1	1	1
LEFTMOST BIT EXCLUDING SIGN	2	2	2	2
LENGTH EXCLUDING SIGN	59	31	35	35
NEGATIVE REPRESENTATION	1	2		2
EXPONENT				
SIGN BIT POSITION				1
LEFTMOST BIT EXCLUDING SIGN				2
LENGTH EXCLUDING SIGN				7
NEGATIVE REPRESENTATION				2
BASE		16		2
FRACTION OR MANTISSA				
SIGN BIT POSITION	1	1	1	9
LEFTMOST BIT EXCLUDING SIGN				10
LENGTH EXCLUDING SIGN		24		27
NEGATIVE REPRESENTATION		SM		2
CHARACTERISTIC				
LEFTMOST BIT EXCLUDING SIGN		2		
LENGTH EXCLUDING SIGN		7		
EXCESS		64		

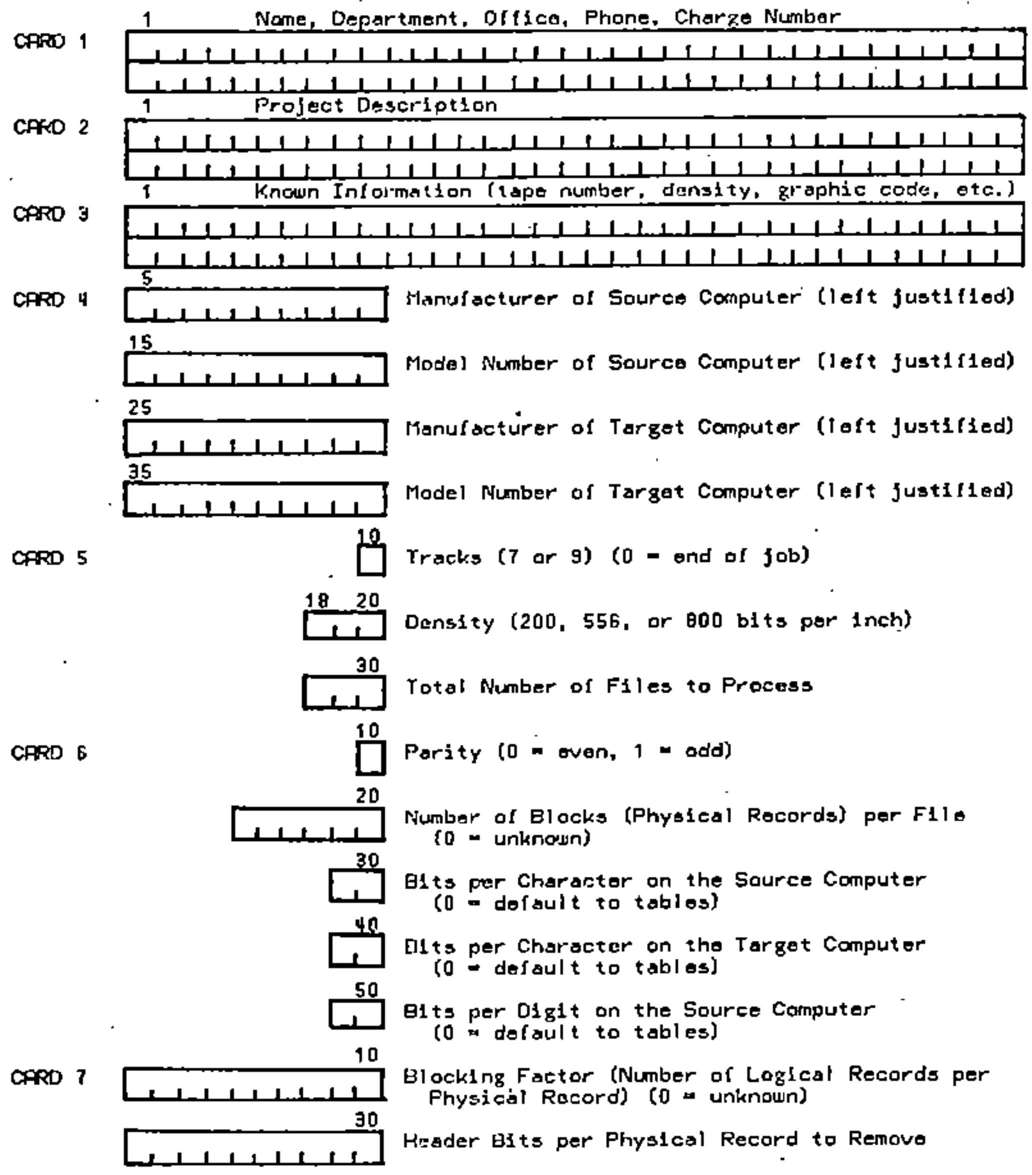
SOURCE AND TARGET DATA DESCRIPTION

The input data necessary to describe the data description of the source tape and the data description of the target tape may be found in the job, file and record descriptor data sheet (see Figure 5.2) and in the field descriptor data sheet (see Figure 5.3).

MEMORY ALLOCATION AND TIMING

The program compiles on the CDC 6500 in less than 62,000 octal words and executes in less than 52,000 octal words.

Timing is a function of the size of each physical record, the number of physical records to be processed on the tape and the types of transformation to be done. For large physical records of approximately 5000 6-bit characters with a straight 6-bit graphic transformation it will execute as high as 5000 characters per second. For physical records of 80 characters with straight 6-bit graphic transformation it will execute at approximately 1500 characters per second. More complicated data descriptions involving blocking factors, leading bits to remove etc. will run at about 1000 characters per second. Timing for a single floating



CARD 1

1 Name, Department, Office, Phone, Charge Number

CARD 2

1 Project Description

CARD 3

1 Known Information (tape number, density, graphic code, etc.)

CARD 4

5 Manufacturer of Source Computer (left justified)

15 Model Number of Source Computer (left justified)

25 Manufacturer of Target Computer (left justified)

35 Model Number of Target Computer (left justified)

CARD 5

10 Tracks (7 or 9) (0 = end of job)

18 20 Density (200, 556, or 800 bits per inch)

30 Total Number of Files to Process

CARD 6

10 Parity (0 = even, 1 = odd)

20 Number of Blocks (Physical Records) per File (0 = unknown)

30 Bits per Character on the Source Computer (0 = default to tables)

40 Bits per Character on the Target Computer (0 = default to tables)

50 Bits per Digit on the Source Computer (0 = default to tables)

CARD 7

10 Blocking Factor (Number of Logical Records per Physical Record) (0 = unknown)

30 Header Bits per Physical Record to Remove

Figure 5.2 Job, Tape, File and Record Descriptor Data Sheet

CARDS 8...n

2 4	Beginning Field Number (0 = last field descriptor)
6 8	Terminating Field Number
10 15	Number of Units to Transform
17 26	Type of Units (left justified)
30	Source Data Type
34	Target Data Type
38	Target Value to be Used Later as a Repeat Counter (0 = No, 1 = Yes)
40 42	Repeat Field Descriptor Based on Value Found in Field K (0 = no repeat, K = the field number)
45	Concatenate Last Target Field With First Target Field on Next Field Descriptor (0 = No, 1 = Yes)
48	Output on Completion of This Field Descriptor (0 = No, 1 = Yes)

Figure 5.3 Field Descriptor Data Sheet.

point number is approximately equivalent to straight transformation of 10 characters.

Exclusive of initialization time (about eight seconds) and tape read time (a variable that ranges between one percent and twenty five percent of the total execution time), the program will transform between 2500 and 3000 characters per second for an average run.

IMPLEMENTATION CONSIDERATIONS

The original program was developed on the CDC 6500. No special design effort was made to take advantage of this computer however. In designing the system, there were several ways to implement a procedure. Each of the major implementation considerations will now be discussed.

Bit Extraction

a) shifting routines - The normal left shift and circular left shift with zero fill could have been replaced by other shifting routines that could accomplish the same end result with some program modifications. Since there are no guaranteed shifting instructions that exist on all computers, it is not possible to choose a universal subset. On a particular computer, the implementations of these two shifting routines may not be optimum in either storage or execution time.

b) bit testing routine - Although this feature could have been implemented by shifting, masking and non zero testing, a bit testing routine was used.

c) masking routines - The OR and AND capability were implemented as functions. In CDC 6500 Fortran, the .OR. and .AND. can be used for non-logical types. However all references are in a function form so that the OR functions and the AND function can be changed by an installation to use other routines.

Error Messages

The majority of the error messages are located in a single subroutine (MESAGE). This is done to isolate the format statements since they can be machine dependent. However, it was felt necessary to let the tape read subroutine (TREED) handle its own machine dependent messages.

Limitations and Assumptions

If the source computer has a word length longer than the target computer, the fixed point data is examined to determine if it will exceed the fixed point length of the target computer. If the length is exceeded, the field is flagged and the job is terminated. It is also assumed that the binary point is to the right of the rightmost bit. The user can circumvent these problems by specifying the input field as more than one straight binary target field and then perform the transformation himself. An example of this would be converting from fixed point on the CDC 6500 (60-bit words) to fixed point on an IBM 370 (32-bit words).

Spanning across physical files is not allowed. It is assumed that if this is necessary, the user can concatenate his files in advance.

Up to 100 field descriptor cards are allowed, the maximum field number is 999 and the maximum number of 6-bit characters per physical record is 15000. If a field descriptor card contains both concatenation and output options, the output option is not exercised. If the physical record count is not zero, upon completion the source tape will be positioned at the end of the last record requested.

In the current production version of the tape transformation program, spanning across physical records is not allowed, sentinel checking has not been implemented, flow control has not been implemented, multiple reel jobs are not allowed, tape number requests are initiated with job control cards, the source computer must be manually selected and the graphic tables must be manually sorted prior to a production run.

PROGRAM VERIFICATION

As the program was developed, test cases were run to verify the different options. Included in these test cases were nineteen of the twenty cases discussed in Chapter 3 (see Table 3.11). Features necessary for case D1 have not been implemented yet. These twenty test cases can hest be described in further detail by examining their job, tape, file and record descriptors as well as their field descriptors (see Tables 5.6 and 5.7). With minor modifications to the RITE subroutine, case I1 verified the possibility of using the program to create a source tape compatible to the target computer when the target system was known in advance.

Table 5.6 Job, Tape, File and Record Descriptors for Test Cases

	CASE						
	C1	D1	H1	I1	I2	I3	I4
SOURCE COMPUTER	CDC 6500	PDP 11/45	HIS 6000	IBM 360	IBM 360	IBM 370/155	IBM 360/67
TARGET COMPUTER	IBM 360	CDC 6500	CDC 6500	CDC 6500	CDC 6500	CDC 6500	CDC 6500
TOTAL NUMBER OF REELS	2	1	1	1	1	1	1
NUMBER OF TRACKS ON ORIGINAL SOURCE TAPE	7	7	7	9	9	7	9
DENSITY (BITS PER INCH) CDC 6500 INPUT TAPE	800	800	556	800	800	800	800
PARITY	ODD	ODD	EVEN	ODD	EVEN	EVEN	ODD
NUMBER OF PHYSICAL FILES	14	2	1	1	1	1	24
SOURCE BITS PER CHARACTER	6	12	6	8	6	6	8
TARGET BITS PER CHARACTER	8	6	6	6	6	6	6
SOURCE BITS PER DIGIT						4	
LOGICAL RECORDS SPAN ACROSS PHYSICAL RECORDS	NO	NO	NO	NO	NO	NO	NO
BLOCKING FACTOR	30	6.4	1	89	1	2	30
HEADER CHARACTERS TO REMOVE PER PHYSICAL RECORD	NO	NO	NO	NO	YES	NO	NO
LARGEST PHYSICAL RECORD (IN SOURCE CHARACTERS)	2400	512	84	7120	806	960	2400

Table 5.6, cont.

	CASE						
	I5	I6	I7	I8	I9	I10	I11
SOURCE COMPUTER	IBM 360	IBM 7074	IBM	IBM 370/165	IBM 370/165	IBM 360	IBM 360/67
TARGET COMPUTER	CDC 6500	CDC 6500	CDC 6500	CDC 6500	CDC 6500	CDC 6500	CDC 6500
TOTAL NUMBER OF REELS	1	3	2	1	1	1	1
NUMBER OF TRACKS ON ORIGINAL SOURCE TAPE	9	7	7	9	9	7	7
DENSITY (BITS PER INCH) CDC 6500 INPUT TAPE	800	556	800	800	800	556	800
PARITY	ODD	EVEN	EVEN	ODD	ODD	EVEN	EVEN
NUMBER OF PHYSICAL FILES	4	2	2	1	1	1	4
SOURCE BITS PER CHARACTER	8	6	6	8		6	6
TARGET BITS PER CHARACTER	6	6	6	6	6	6	6
SOURCE BITS PER DIGIT		4					
LOGICAL RECORDS SPAN ACROSS PHYSICAL RECORDS	NO	YES	YES	YES	NO	NO	NO
BLOCKING FACTOR	1	1/2	1/2	1/2	199	1	30
HEADER CHARACTERS TO REMOVE PER PHYSICAL RECORD	NO	NO	NO	YES	YES	NO	NO
LARGEST PHYSICAL RECORD (IN SOURCE CHARACTERS)	80	911	5130	80	800	80	2400

Table 5.6, cont.

	CASE					
	M1	M2	S1	U1	U2	U3
SOURCE COMPUTER	MODCOMP III	MODCOMP III	SIEMENS	UNIVAC 1108	UNIVAC 1108	UNIVAC 1106
TARGET COMPUTER	CDC 6500	CDC 6500	CDC 6500	CDC. 6500	CDC 6500	CDC 6500
TOTAL NUMBER OF REELS	1	1	1	1	1	1
NUMBER OF TRACKS ON ORIGINAL SOURCE TAPE	7	7	9	7	9	7
DENSITY (BITS PER INCH) CDC 6500 INPUT TAPE	800	800	800	800	800	556
PARITY	ODD	ODD	ODD	ODD	ODD	EVEN
NUMBER OF PHYSICAL FILES	1	1	1	1	1	1
SOURCE BITS PER CHARACTER	7		8	6	6	6
TARGET BITS PER CHARACTER	6		6	6	6	6
SOURCE BITS PER DIGIT						
LOGICAL RECORDS SPAN ACROSS PHYSICAL RECORDS	NO		NO	YES	YES	YES
BLOCKING FACTOR	40		1	VARIABLE	VARIABLE	VARIABLE
HEADER CHARACTERS TO REMOVE PER PHYSICAL RECORD	NO		NO	YES	YES	YES
LARGEST PHYSICAL RECORD (IN SOURCE CHARACTERS)	80		80	10764	1344	1344

Table 5.7 Field Descriptors for Test Cases

	CASE									
	C1	D1	H1	I1	I2	I3	I4	I5	I6	I7
UNIT TYPE										
BITS		X								
CHARACTERS	X	X	X	X	X	X	X	X	X	X
WORDS		X								
FIELD TYPE										
NULL		X			X					
STRAIGHT BINARY										
FIXED POINT		IN								
FLOATING POINT										
NUMERIC DECIMAL									IN	
PACKED DECIMAL						IN				
DISPLAY (CDC 6500)	IN	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT
BCDIC (ALTERNATE)										IN
BCDIC (PRINCIPLE)					IN	IN				
BCD BASIC									IN	
BCD BASIC + 7			IN							
BCD (UNIVAC 1108)										
FIELDATA (UNIVAC 1108)										
ASCII (7-BIT)										
EBCDIC	OUT			IN			IN	IN		
OTHER (12-BIT ASCII)		IN								
VARIABLE LENGTH LOGICAL RECORDS										
CONCATENATED OUTPUT FIELDS	X	X							X	X
OUTPUT CONTROL	X	X							X	X
SENTINEL CONTROL										

Table 5.7, cont.

	CASE									
	I8	I9	I10	I11	M1	M2	S1	U1	U2	U3
UNIT TYPE										
BITS	X	X			X			X	X	X
CHARACTERS	X		X	X	X		X	X	X	X
WORDS		X								
FIELD TYPE										
NULL	X	X			X			X	X	X
STRAIGHT BINARY								IN	IN	IN
FIXED POINT										
FLOATING POINT		IN								
NUMERIC DECIMAL										
PACKED DECIMAL										
DISPLAY (CDC 6500)	OUT	OUT	OUT	OUT	OUT		OUT	OUT	OUT	OUT
BCDIC (ALTERNATE)			IN	IN						
BCDIC (PRINCIPLE)										
BCD BASIC										
BCD BASIC + 7										
BCD (UNIVAC 1108)										IN
FIELDATA (UNIVAC 1108)								IN	IN	OUT/IN
ASCII (7-BIT)					IN					
EBCDIC	IN						IN			
OTHER (12-BIT ASCII)										
VARIABLE LENGTH LOGICAL RECORDS								X	X	X
CONCATENATED OUTPUT FIELDS	X				X			X	X	X
OUTPUT CONTROL	X	X		X				X	X	X
SENTINEL CONTROL		X						X	X	X

MACHINE INDEPENDENCE

Since the major issue of this thesis is "portability", it was felt that the tape transformation program should be as portable as possible.

The program, excluding octal and entry, meets the ANSI standards as specified under the Minnesota Fortran Compiler(MNF). It is currently being compiled, with minor modifications, under both Fortran G on an IBM 370/155 and the Fortran available on the PDP 11/45. The machine dependent portions of the program are identified and separated from the main program. The most flagrant violations of machine independence are in the dimension and common statements that assume 10 characters per word and the capability to read and write octal numbers. With minor modifications, a proper read package and bit manipulation facilities, this program can successfully run on another computer with a Fortran compiler.

In order to compile the tape transformation program on another computer, the following steps should be taken:

1) Run all subroutines and the main program through the Fortran compiler of the target computer.
2) Change dimension and common statements to reflect size of arrays necessary for sufficient characters.

3) Change format, read and write statements to reflect maximum number of characters per word.
4) Recode read (TREED) and write (RITE) subroutines to reflect new computer dependencies.
5) Recode functions to allow bit shifting capabilities and bit testing.
6) Recode all remaining machine dependent subroutines.
7) Correct all errors.

CHAPTER 6

OPERATIONAL GUIDELINES FOR MAGNETIC TAPE INTERCHANGE

GENERAL COMMENTS

The standards and guidelines for magnetic tape interchange that exist were discussed in Chapter 3. As has been noted, the guidelines and standards are not very definitive and for this reason one may expect to receive magnetic tapes are in various formats. What is needed then are realistic guidelines to follow when generating a source tape to be used elsewhere and then some techniques to transform the source tape to a target tape once it arrives.

GENERATION OF ACCEPTABLE TAPES

Excluding physical constraints, the first step in generating acceptable tapes that will be readable at other installations is being able to properly describe the data that resides on the tape.

Tape Data Description

The necessary parameters to describe the contents of a tape must be sent with the tape. One can never assume that the recipient of the tape is completely knowledgeable regarding its contents and format. To alleviate this problem, it is suggested that the following parameters be sent.

Manufacturer and model number of the source computer
Manufacturer and model number of the tape drive
Total number of reels
Number of tracks on each reel
Density of each real (in bits per lineal inch)
Total number of physical files on each reel
Total number of logical files in each physical file
Parity of each logical and physical file
Number of physical records in each physical file
Number of logical records per physical record (blocking factor)
Maximum number of characters per physical record
Number of bits per character or byte
Number of bits per digit
Number of bits per word
Data description of each field
Contents of internal labels if they are used

Additional Requirements

In addition to the magnetic tape and a description of its contents, a listing, graphic tables, tape control cards, and decompression algorithms should be sent. External labels should be placed on the tape reel and a convention for multiple reel jobs should be furnished.

Listing

A listing of the contents of the tape should be sent along with the tape. Regardless of the expense of producing and mailing this listing, it provides the recipient of the data with what he should find on the tape. The absence of a listing may result in excessive time consumed in transforming the tape since the recipient has only a vauge idea as to its contents. The actual contents of the tape may not necessarily conform to the listing due to decompressed data or the utility routine that placed the data on the tape.

Graphic Tables

The graphic symbols of the source computer and their corresponding numeric code should also accompany the tape. This allows the recipient to determine which graphic symbols are different between the source computer and the target computer. It also allows him to determine if the numeric code for a specific graphic symbol is the same on both the source and target computers. There are no standard graphic tables in use today since most installations make at least one minor change. These tables generally exist in programming or installation manuals that are available at the site of the source computer.

Tape Control Cards

A listing of the control cards that generated the source tape should also be sent. Control cards are very useful even if the source computer and its operating system are very different from those of the target computer. They provide a frame of reference in which further questions may be asked if needed. For example, IBM 360/370 job control cards provide valuable information about the contents of a tape even to non IBM installations.

Consider the two following examples:

```
1) DCB=(RECFM=F,LRECL=5130,BLKSIZE=5130),LABEL=(,NL)
2) DCB=(RECFM=FB,LRECL=80,BLKSIZE=2400,DEN=2,TRTCH=ET)
```

In example 1, the logical record size (LRECL) is 5130 characters, blocksize (BLKSIZE) is 5130 characters and there are no labels (NL). Therefore we have an unlabelled tape with a blocking factor of one with 5130 characters per physical record. In example 2, the logical record size is 80 characters, blocksize is 2400 characters, density (DEN) is 800 bpi and even parity with translation to provide 8 to 6 bit translation (ET). Therefore the tape has a blocking factor of 30 with 2400 6-bit characters per physical record at 800 bpi.

Decompression Algorithm

If the tape must be sent in a compressed form, the algorithm for decompression should also be sent since without it the user will not be able to extract the data. The algorithm describes exactly how to extract the data from the tape. If the decompression algorithm can not be properly described, this is a very strong indication that the tape should not be sent in this format.

Multiple Reels

On multiple reel jobs, a physical end of file should be written before the end of tape reflective mark. In most cases, the current physical record is completed on writing after an end of tape reflective spot is reached. On reading however, it is left to the user to know how many physical records exist on his reel of tape. All systems do not handle the end of tape reel reflective condition in the same way. For example, the CDC 604 tape drives have difficulty reading data within four inches of the reflective mark. One solution to this problem is always to have an end of file mark at least one foot before the end of tape reflective mark.

External Tape Labels

A tape label should be placed on the reel of tape (not the container or cannister) containing as much information as possible. This label should contain pertinent information about the characteristics of each file such as the number of tracks, density, number of files, parity, graphic code, blocking factors and number of records in each file.

STEPS FOR A SUCCESSFUL TAPE TRANSFORMATION

The user must always be aware that tapes tend to be machine dependent as well as operating system dependent. As we find out more about the tape, we are able to proceed further with the transformation. Thus the process of transformation is iterative in nature.

There are two programs that are necessary if the user is to successfully transform a tape. He needs a universal tape dump program which may or may not be present in the utility routines in the operating system. He also needs a tape transformation program. In addition, he needs to be aware of the transformation process, manual techniques for tape transformation and possible causes of errors that may occur when reading magnetic tapes. These topics are now discussed.

Universal Tape Dump

For tapes whose contents are unknown, a tape dump is used to establish density, parity and the exact contents of the tape prior to its transformation. It can also be used to supply a first guess as to the contents of the tape. The user must also have the capability of being able to dump the original tape even after the transformation has occurred. With a tape dump it is often possible to locate portions of invalid data due to bit dropouts or minor tape malfunctions. When supplied with this additional information the input data description can be modified to help produce valid data.

Tape Transformation Program

This program should consist of the topics covered in chapter 4. An implementation was discussed in Chapter 5.

The Transformation Process

Before starting the process, the user should gather all information available to him. This hopefully should consist of the items that have been mentioned earlier in this chapter under the generation of acceptable tapes.

There is a 7 step man-machine process that generally takes place during the transformation of a tape from one that is acceptable for the source computer to one that is acceptable to the target computer. Assuming that the tape is physically compatible as to the number of tracks and the width of the tape that are available on the target machine, these 7 steps are:

1) Check the density of each source tape and the parity of each file.

2) Count the number of physical records per file. Verify that all of the files are there that are supposed to be there. Identify possible tape errors. A count of words, bytes or characters per physical record will often give an indication as to the

contents of the tape.

3) Dump the first few physical records of each file in either octal, decimal or hexadecimal to verify the suggested transformation code. If the output of step 2 looks different than was expected, dump the entire contents of each file. It is better to do this now than to waste time later.

4) Make a short test run using the suspected transformation code to verify that it is correct. If it is not correct, use another transformation code or modify the current one and repeat this step.

5) Make a complete run to find all errors. Correct these errors if possible. If no errors occur then skip to step 7.

6) Make a final run.

7) Verify that the target tape was generated correctly.

Steps 1 through 3 above can be accomplished using a universal tape dump program. Steps 4 through 6 require a tape transformation program. Step 7 requires an utility routine capable of verifying that the target tape was created without physical errors.

Manual Techniques for Tape Transformation

If the contents of the tape are unknown or partially unknown, there are several techniques that may be used to help ascertain the graphic code.

1) Dump the tape and look for occurrences of blanks. If the contents of the tape are card images, strings of blanks may occur. This gives a hint as to which possible code to try.

   examples of blank strings in octal:

   5555555555 implies CDC 6500 Display code
   6060606060 implies internal BCD
   2020202020 implies external BCD
   0505050505 implies Univac 1108 Fieldata
   200401002004 0100 implies EBCDIC

2) Look for other characters such as a string of asterisks, dollar signs, minus signs, etc. that are repeated. Many programmers use these characters to separate sections of code.

   examples of asterisk strings in octal:

   4747474747 implies CDC 6500 Display code
   5454545454 implies BCD (external or internal)
   5050505050 implies Univac 1108 Fieldata
   270561342705 6134 implies EBCDIC

examples of dollar sign strings in octal:

5353535353 implies CDC 6500 Display code or BCD

4747474747 implies Univac 1108 Fieldata

2665553326655533 implies EBCDIC

examples of minus sign strings in octal:

4646464646 implies CDC 6500 Display code

4040404040 implies BCD (external or internal)

4141414141 implies Univac 1108 Fieldata

3006014030060140 implies EBCDIC

3) Look for expected patterns. In particular, if we know we are trying to transform a Fortran program, there should be several card images that have a C in column one. Thus we can take this numeric code in column one and back into the graphic symbol with the help of the master graphic table.

example of a "C" in octal:

03 implies CDC 6500 Display code

23 implies internal BCD

63 implies external BCD

10 implies Univac 1108 Fieldata

303 implies EBCDIC

4) If the tape is compressed, look for special characters that are used to replace sequences of characters. Examples include replacing 5 blanks by 3% or replacing 9 zeroes by 7# where % and # are special characters used for compression flags.

5) Telephone the person responsible for generating the tape for additional information.

Once the code has been ascertained, it may be necessary to identify the operating system that produced it.

1) Look for the control words or symbols at the start of each file or record.

2) Look for repeating codes that appear as end of line sentinels, end of record sentinels and end of file sentinels.

examples in octal:

a) 1632 in the rightmost 12 bits of a word implies a CDC 6500 SCOPE tape

b) 0000 in the rightmost 12 bits of a word implies a CDC 6500 MACE tape

If the tape appears to be an unknown BCD, the following steps should be useful.

1) Use BCD Basic as the source code. This will transform the numerics 0 to 9 and the letters A to Z.

2) If BCD Basic looks correct, assume that period, blank, minus, slash, asterisk, dollar sign and comma will transform correctly. Use Basic BCD + 7 as the source code.

3) If we have a Fortran program, it should be obvious after the first run which are the opening and closing parenthesis, equals and plus sign. The remaining characters may not be as easy.

4) After as many characters have been ascertained as possible, phone the sender and ask him to look on his listing to see which graphic symbol is used for the missing numdric codes.

If the tape is not compressed and it is not graphic, it could be either straight binary, fixed point or floating point. If the range of data is known, all transformations can be tried as a first guess before telephoning the person responsible for generating the source tape.

Magnetic Tape Errors

If persistent tape errors occur on reading the source tape, it can be either the source tape or the tape drive that is malfunctioning on the target computer. If it is suspected that the source tape is acceptable, the following questions should be asked:

1) How often are the tape drives cleaned?

2) Is it possible that the tape occupying the tape drive immediately preceeding yours was a dirty tape?

3) How reliable are the tape drives? Have other users been having problems with the tape drives? Is there a specific tape drive that has been giving problems?

If the source tape is suspect, it may be due to one of the following problems.

1) The bits on the tape are skewed. This happens when the write head on the tape drive is not perpendicular to the magnetic tape.
2) The recorded density is not within specifications. This can be checked by developing the tape.
3) The tape is of poor quality. It may be old, dirty, have creases in it or in general a bad tape.

CHAPTER 7

CONCLUSIONS AND FURTHER RESEARCH AREAS

CONCLUSIONS

A procedure has been described which will transform a magnetic tape generated on a source computer to one that is acceptable on a target computer. The design has been of a modular form so that improvements can be easily made. The implementation has shown that tape transformation can be removed from requiring the user to revert to the level of assembly language programming. Aside from transforming tapes, the implementation has been useful in identifying and locating special graphic symbols. In particular, machine dependent programming has been observed when the only occurrences of a graphic symbol appear only in comment statements in Fortran programs.

The major contributions that have been made are:

1. Developing a methodology and a program suitable as a tool for users so that they can transform their own tapes.

2. Developing a data description that allows the user to define the contents of his magnetic tape.

3. Placing the transformation of tapes at a higher level than assembly language programming.

4. Identifying machine dependent and machine independent sections of the transformation process.

5. Providing realistic guidelines for magnetic tape interchange.

EXTENSIONS AND FURTHER RESEARCH

There are several instances where transforming a magnetic tape using remote batch facilities takes a considerable amount of elapsed time before the transformation is complete. This suggests extending the approach described in Chapter 4 to include an interactive mode. The user could then be presented with and examine more possibilities which in turn could result in faster transformation of the tape.

The approach discussed in Chapter 4 is optimal neither in time nor in space. If it were possible to properly describe the data on the magnetic tape, after a limited number of iterations, the resulting program could then be passed through a reorganizer that would produce more efficient code. Nylin (1972) has obtained some useful results in this area that can be used. The savings of processing time on very large jobs would more than justify this extension.

The model has been presented to allow implementation of an algorithm on a target computer to transform a source tape to a target tape. The model could be expanded to include a host computer for the algorithm. Thus, a source tape could be transformed on a host computer into a suitable target tape for the target computer when the target system was well defined.

There are type limitations imposed on the approach in Chapter 4 that could be relaxed. The user may wish to transform straight binary fields to fixed point fields or floating point fields, etc. Allowing these additional type transformations extends the usefulness of the approach.

A much larger task would be to expand the model and the approach to allow for data description on non-sequential devices such as disks or drums. This infers allowing an additional dimension to be referenced and understanding the techniques used to place data on these devices. The impact of these devices on magnetic tape has already been shown but the impact of drums on disks and vice versa still remains an unknown.

LIST OF REFERENCES

LIST OF REFERENCES

ANSI
"Recorded Magnetic Tape for Information Interchange (1600 CPI, Phase Encoded)"
Communications of the ACM
Volume 13/Number 11/November, 1970
pp 679-685

ANSI
"Recorded Magnetic Tape for Information Interchange (200 CPI, NRZI)"
Communications of the ACM
Volume 10/Number 11/November, 1967
pp 730-737

ANSI
"Magnetic Tape Labels for Information Interchange"
Communications of the ACM
Volume 10/Number 11/November, 1967
pp 737-743

CDC
"8-Bit Subroutine Reference Manual"
Working Copy No. 60359400 A
1972
Control Data Corporation

Clamons, E. H.
"Character Codes: Who Needs Them?"
Honeywell Computer Journal
Volume 5/Number 3/1971
pp 143-148

Climenson, W. Douglas
"File Organization and Search Techniques"
In: Annual Review of Information Science and Technology
Volume 1
Edited by Cuadra, Carlos A.
John Wiley (Interscience) 1966
pp 107-135

CODASYL
"A Survey of Generalized Data Base Management Systems"
CODASYL Systems Committee
Available from The Association for Computing Machinery
May 1969

CODASYL
"Feature Analysis of Generalized Data Base Management Systems"
CODASYL Systems Committee
Available from the Association for Computing Machinery
May 1971

DBTG
"Data Base Task Group Report to the CODASYL Programming Language Committee"
CODASYL Data Base Task Group
Available from The Association for Computing Machinery
October 1969

DBTG
"Codasyl Data Base Task Group Report"
CODASYL Data Base Task Group
Available from the Association for Computing Machinery
April 1971

Diebold
"Data Management Software"
Diebold Research Program
Document Number T24
September 1969

FIPS PUB 1
"USA Standard Code for Information Interchange"
Federal Information Processing Standards Publications
USAS X3.4-1968
U.S. Department of Commerce, National Bureau of Standards
October 10, 1968
15 pages

FIPS PUB 20
"Guidelines for Describing Information Interchange Formats"
Federal Information Processing Standards Publications
SD catalog No C 13.52:20
U.S. Department of Commerce, National Bureau of Standards
March 1,1972
11 pages

French, A., J. Ramirez, H. Solow, N.S. Prywes
"Design of the Data Description Language Processor"
Moore School Report No. 72-19
University of Pennsylvania
December, 1971

Fry, J. P. et al.
"Data Management Systems Survey"
MTP-329, Rev. 1
The MITRE Corporation
May 1969

Fry, James P., Randall L. Frank and Ernest A. Hershey
"A Developmental Model for Data Translation"
ISDOS Working Paper #60
Department of Industrial and Operations Engineering
University of Michigan, Ann Arbor, Michigan
August 1972

Gosden, John A.
"Software Compatibility: what was promised, what we have, what we need"
Proceedings Fall Joint Computer Conference 1968
pp 81-87

Guide-Share
"Data Base Management System Requirements"
Joint Share and Guide Data Base Requirements Group
Available from Share
November 11, 1970

IBM
"IBM System/360: Planning for the Use of Information Interchange Standards OS DOS TOS"
Order Number GC 28-6756-0
International Business Machines
June, 1970

IBM
"OS Tape Labels"
Release 21
Order Number GC28-6680-4
International Business Machines Corporation
February 1972

Mealy, George H., et al.
"Program Transferability Study"
(AD 678 589)
November 1968

MIDMS
"Machine Independent Data Management System (MIDMS) - Users Reference Manual"
Defense Intelligence Agency Manual No 65-9-9
Headquarters, Defense Intelligence Agency
Washington, D. C. 20301
March 12, 1971

Minker, Jack and Jerome Sable
"File Organization and Data Management"
In: Annual Review of Information Science and Technology
Volume 2
Edited by Cuadra, Carlos A.
John Wiley (Interscience) 1967
pp 123-160

Minker, Jack
"Generalized Data Management Systems-Some Perspectives"
Technical Report 69-101
University of Maryland
Computer Science Center
December 1969

Nunamaker, J. F. Jr., D, E. Swenson and A. B. Whinston
"Specifications for the Development of a Generalized Data Base Planning System"
National Computer Conference, 1973
pp 259-270

Nylin, William Carl,Jr.
"Structural Reorganization of Multipass Computer Systems"
Ph.D.Thesis
Purdue University, Lafayette, Indiana
June, 1972

Ross, H. McG.
"The ISO Character Code"
The Computer Journal
Volume/Number3/October 1964
pp 197-202

Ruth, Stephen S. and Paul J. Kreutzer
"Data Compression for Large Business Files"
Datamation, September 1972
pp 62-66

Sattley, Kirk, Robert Milstein and Stephen Warshall
"On Program Transferability"
Massachusetts Computer Associates
CA-7011-2411
November 24, 1970

Senko, Michael E.
"File Organization and Management Information Systems"
In: Annual Review of Information Science and Technology
Volume 4
Edited by Cuadra, Carlos A.
Encyclopedia Britannica 1969
pp 111-143

Shoffner, Ralph M.
"The Organization, Maintenance, and Search of Machine Files"
In: Annual Review of Information Science and Technology
Volume 3
Edited by Cuadra, Carlos A.
Encyclopedia Britannica 1968
pp 137-167

Sibley, Edgar H. (Director)
"Design Specifications of a Prototype Data Translator"
Data Translation Project
University of Michigan
Department of Industrial and Operations Engineering
October, 1972

SIGFIDET
"Record of the 1970 ACM SICFIDET Workshop on Data Description and Access"
Rice University
Available from the Association for Computing Machinery
November 15-16, 1970

SIGFIDET
"Proceedings of 1971 ACM-SIGFIDET Workshop
Data Description, Access and Control"
San Diego, California
Edited by E. F. Codd and A. L. Dean
Available from the Association for Computing Machinery
November 11-12, 1971

SIGFIDET
"Proceedings of 1972 ACM-SIGFIDET Workshop
Data Description, Access and Control" Denver, Colorado
Edited by A. L. Dean
Available from the Association for Computing Machinery
November 29, 30 - December 1, 1972

Smith, Diane Pirog
"An Approach to Data Description and Conversion"
Moore School Report No. 72-20
University of Pennsylvania
December, 1971