A GENERAL PURPOSE PROGRAMMING SYSTEM FOR RANDOM ACCESS MEMORIES

C. W. Bachman
General Electric Company
Phoenix, Arizona
and
S. B. Williams
General Electric Company
New York, New York

I. INTRODUCTION

During the past ten years, information processing technology has made significant advances in many directions. Faster, less expensive, more flexible *hardware* has been continually announced by the various computer manufacturers. In the *software* area, the FORTRAN, ALGOL, and COBOL languages have been developed and improved and more efficient compilers are now available. *Applications* now include the complete spectrum ranging from free-standing analytical programs to large complex information processing systems.

Computers have been applied to business information processing problems with varying degrees of success. Many accounting operations and facets of historical record-keeping have been mechanized with proven time, cost, and accuracy benefits. Those types of business operations dealing with planning and control (or command and control if you are part of a military establishment) are receiving considerable attention from the mechanization standpoint. While many mechanization attempts have been made in this area, the proven successes are few. To some extent this can be attributed to the greater complexity of these classes of problems and the fact that information must be stored, retrieved, communicated, and processed concurrent with the flow of orders and materials.

The information processing field seems to be moving exponentially in the direction of "real time" and total or highly integrated information systems. This movement has been accelerated by the introduction of larger, faster, and more economical mass random access memory devices coupled with faster computers and better communication equipment. These new facilities offer the information system designer a new opportunity 1) to organize his information files with minimum duplication and redundancy, 2) to provide a better man-machine interface by giving people quick access to information, 3) to store, retrieve, and process information when the need arises rather than when the computer schedules dictate, 4) to provide a single data base for many applications as opposed to the arbitrary sequencing of single files for each particular application.

Any attempt to exploit the opportunities presented by the new mass memory devices places a high burden on the information system designers and programmers. This is true because it is difficult to structure and organize complex information relationships within the parameters of the mass memory devices. It is also

very difficult to write computer programs to store, maintain, retrieve, and process the complex data. To date there has been little if any software available to facilitate these problems.

The General Electric Company through its Corporate Services has been conducting a continuing research program on the manufacturing control problem since 1956. The decision table and TABSOL techniques resulting from this research work were described to the information processing world in 1960.[1] During the past four years a considerable effort has gone into studying the information requirements for manufacturing control and how the information might be organized and processed more effectively by using mass memory devices. As a result of this work, a new approach has been developed—the Integrated Data Store.

II. THE INTEGRATED DATA STORE—A NEW APPROACH

The purpose of this paper is to introduce the Integrated Data Store, a general purpose programming system for mass random access storage devices. The particular implementation that will be described is now being installed at several General Electric sites using a GE–215 or GE–225 computer. The Integrated Data Store language and functions will be available early in 1965 as extensions to the COBOL compilers for the new GE–400 and 600 series computers. The principles involved, however, are completely general purpose and could be readily adapted to any general purpose computer to which a mass memory device can be attached.

The Integrated Data Store has been designed from a user's point of view by users. Furthermore, it is a product that draws upon the interest and ideas of many General Electric people with vast and diverse experience as users of computers in business. Particular credit is due Homer Carney of the New York Information Processing Center (GE Computer Department) who long served as the senior programmer on the project and Irv Burch and Bill Helgeson of the Internal Automation Operation whose ideas heavily influenced the current organization of the system. Jerry Aman, Ed Dodge, Phil Farmer, John Gallagher, Jane Gilbane, George Hess, Dave Johnson, Dave Lattemore, Ron Pulfer, and Tom Waldron are others who have had a significant impact upon the specification or programming of the system. Many others have been helpful since the beginning of the Integrated Data Store work in 1961.

III. INTEGRATED DATA STORE—ADVANTAGES

The original Integrated Data Store software package was used in mid 1963 to make comparisons against conventional random access programming techniques in systems design effort, programming effort, file utilization, and computer running time. The IDS compared very favorable on all counts. Since that time, further refinements have been made to the software package.

Experience to date using IDS has demonstrated the following advantages:

1. Greater insight and understanding of information relationships.
2. Reduced time and cost to design, program, and test comparable applications.
3. More efficient computer processing.
4. Better data storage unit utilization through redundancy elimination.

IV. INTEGRATED DATA STORE—ORGANIZATION

The IDS can be described best if it is divided into three areas of discussion:

a. Data Organization—Technique for Mass Memory
b. Data and Procedural Language
c. Input/Output Controller

Data Organization refers to the establishment of inter record relationships within the IDS. This association is achieved through the use of chains which provide cross reference linkages between records. These chains provide the integrated force which is implied in the name, "Integrated Data Store."

Data and Procedural Language refers to the definition of records and their chain associations, and the procedural verbs by which these records are stored and retrieved.

The Input/Output Controller refers to the physical manipulation of the mass random access device and the buffering and housekeeping

associated with temporarily storing blocks of data in core memory.

A. *Data Organization.* The record is the major unit of data organization in the Integrated Data Store.

This is a record in the GECOM (General Compiler) and COBOL sense. It contains a set of data fields which collectively describe the event, thing, status, or plan that the record represents. The Integrated Data Store augments these records with additional fields called chain fields which contain the address of other Integrated Data Store records. The chain fields point from one record to the next creating a serial association of records.

This association is constructed according to the data definitions and the executed procedural commands. The chain is the record organization technique used by the IDS for meaningful associations of records.

B. *Data and Procedural Language.* The Integrated Data Store provides its user with the ability and requirement to predefine his records, their data fields, and their chain fields. Once these records and fields have been defined, the user is free to operate upon the records without concern for the physical aspects of input or output, the linking of records into chains, or the protection of the data from erroneous access.

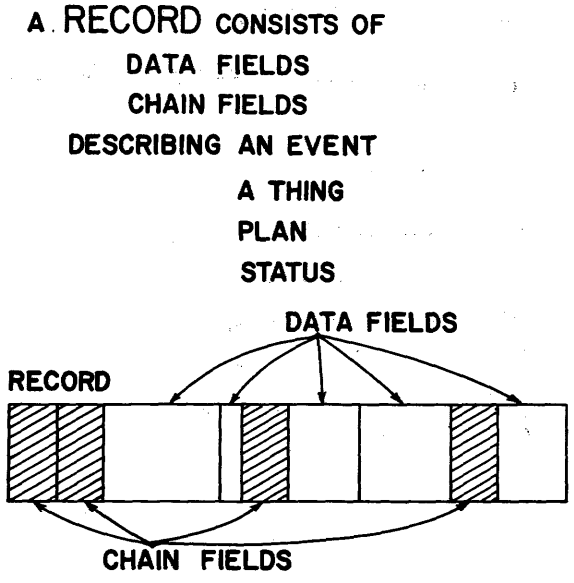

Figure 1. Record Definition.

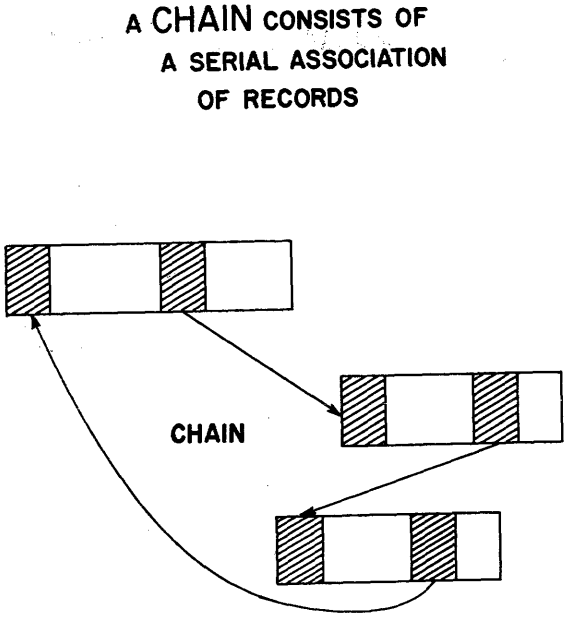

Figure 2. Chain Definition.

The user has four new commands or procedural verbs at his disposal. These verbs provide for the execution of the four basic record processing functions and are complementary to existing COBOL and FORTRAN procedural verbs. These are; "PUT" to store a new record into the file and link it into chains as specified in the data description, "GET" to retrieve a record already in the system, "MODIFY" to change the content of one or several data fields with automatic relinking of chains, if necessary, and "DELETE" to delink a record from its chains and remove it from the file.

C. *Input/Output Controller.* The Input/Output Controller of the IDS controls the data storage device.

It transfers data blocks in and out of core in response to commands to retrieve a specific record, to store a specific record, or to expand or contract a specific record. In order to minimize the data storage device seek and transfer time, an inventory of data blocks is maintained in core memory. These blocks are stored in numerous buffers in core. The number of buffers depends on the amount of space available after the IDS subroutines and the problem solving routine have been loaded. The larger the number of data blocks stored in core, the greater

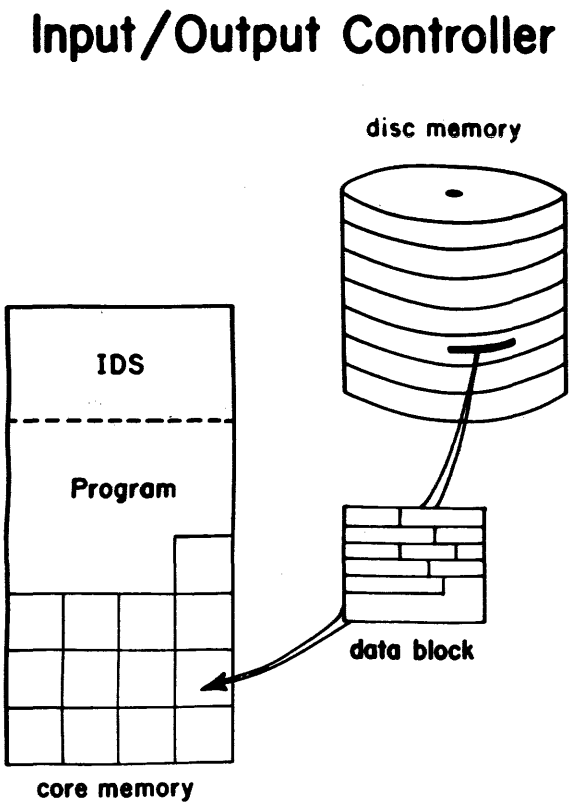

Figure 3. Input/Output Controller.

the possibility that the one needed next will already be in core. To improve the probability of finding the block desired in core, the I/O Controller keeps track of the sequence of block retrieval and utilization and holds the most recently active data blocks in the buffers. Blocks which are not frequently accessed are retired from core to make room as others are called in. The I/O Controller notes which blocks have been modified and writes only the modified blocks back to the data storage unit. The IDS data block manipulation is analogous to the program block "page turning" of the Ferranti Atlas computer.

V. DATA BLOCKS

Looking closer at IDS data blocks, the following characteristics should be observed.

They have a fixed maximum size which is an environmental constant. They consist of one or more data records which collectively represent the actual size of the block. The maximum number of data records is controlled by the size of the maximum block. Every block begins with a block header record. The block header record contains several data fields used by the system. One indicates the space available in the block for additional records, or record expansion within the block. Another indicates whether the block has been altered since retrieval. Still another is a chain field which indicates the address of the first record of a chain of records, all of which randomized to that block.

VI. DATA RECORDS AND FIELDS

The records of the IDS are fixed format, fixed length records in the GECOM, COBOL tradition, i.e. a specific type of record such as a payroll or inventory record has a fixed length and format. Variability in the conventional sense of record length is automatically achieved through the IDS techniques of data structuring. A master record is used with a variable number of detail records.

Records of many different types, each with differing length and format may be used in the system and may be stored within the same

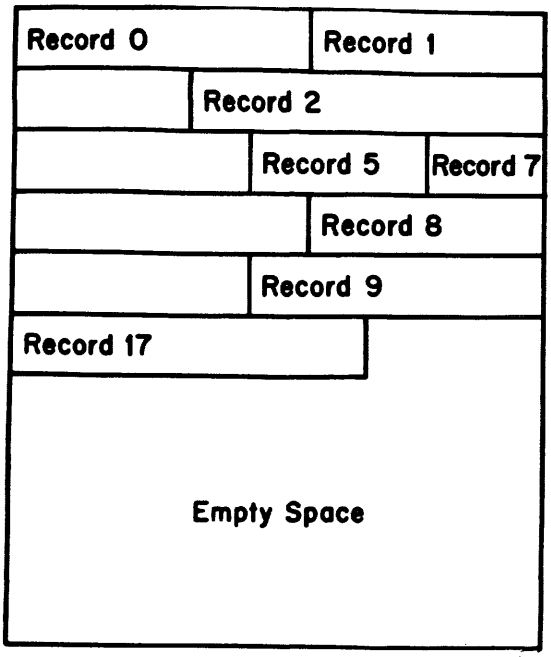

Figure 4. IDS Data Block.

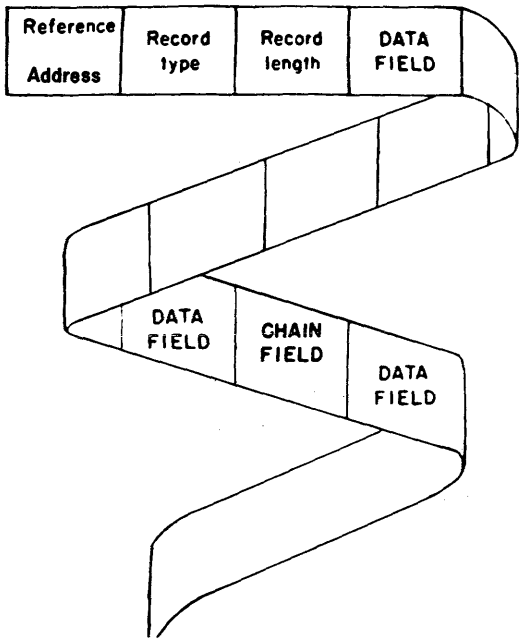

Figure 5. IDS Record Structure.

block. In order that control may be maintained, each record has the same three fields at the very beginning. These fields are the reference code (block number and intra-block record number), the record type, and the record length. The balance of the record consists of data and chain fields in the number and variety to suit the application requirements. Data fields may be defined as being in a logical mode (bits), signed binary numeric mode (one or two words) or an alphanumeric mode (characters). Fields may vary in size from a one bit switch up to many characters for a drawing and part number or a man's name. These fields will be specified by the systems designer.

Chain fields are defined for each chain in which a record participates. Experience in IDS systems indicates that the average record is in only two chains, and an occasional pivotal record in the information integration may be in six or eight chains. There is no upper limit on data or chain fields except that which is provided by the maximum block size. The average record in installations today has been eight to twelve words in total length, with an occasional record type in the forty to sixty word size. These are twenty bit, three character words.

The reference codes in the IDS chain fields are not physical addresses which specify particular discs, tracks and heads. They are more properly described as relative addresses which indicate a relative position in the total environment of mass storage. Therefore, an expansion or contraction of the number or size of the data storage units does not destroy the existing reference codes. It merely changes the mapping function which translates a particular reference code into its disc, track, and head number.

The IDS Records are stored only once in the IDS.

This has three important advantages. First, the additional space required for duplicate records is eliminated, resulting in a reduction in the total storage capacity required. Second, the work of data maintenance is greatly reduced as there is only one record to retrieve and modify. This eliminates the possibility that one of the copies of a record will not be properly modified. As there is only one copy of a record, all users

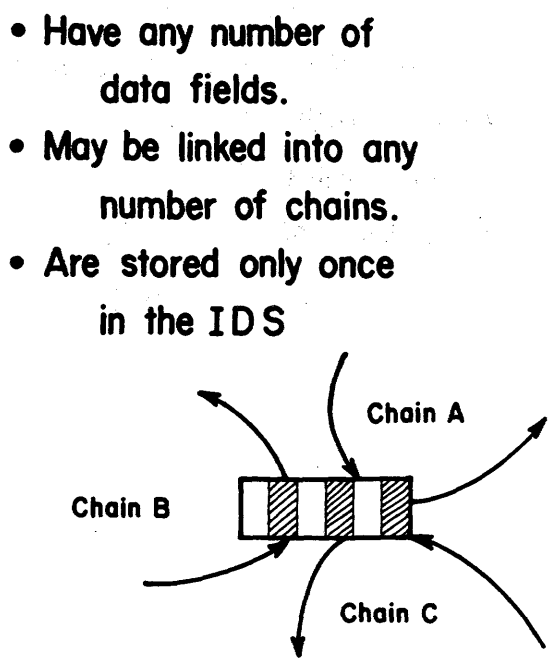

Figure 6. IDS Record/Chain Structure.

have their eyes on it and incorrect information will be quickly spotted and corrected. Finally, all reports, drawn from the file, will be consistent since there is only one set of facts (records).

VII. CHAINS

The IDS chains have several structural aspects which should be emphasized.

Each chain has only one master record. The record type of the master is specified when the chain is defined in the data definitions. Whenever a master record in "PUT" into the IDS, a chain is created which has no details in it. The chain field in the master record stores the reference code of the next record in the chain, which initially is the reference code of the master record itself. As additional records, that are specified as details, are "PUT" into the file, they are linked into the chain. However, the chain always closes back on its master. The position in the chain of a new detail depends on the chain specifications.

- **Have one master record and any number of details.**
- **Link records together in an endless loop.**
- **Associate related records in meaningful sequences.**

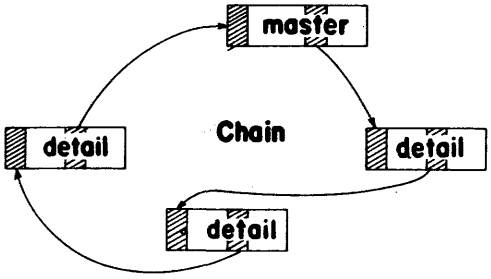

Figure 7. IDS Chain Structure.

It was previously stated that a record may be in any number of chains. Now it is worth expanding this statement to read that a record may serve as a master or detail in any number of chains. The only restraint is that no record may be a detail of itself directly, or through the interaction of several chains.

VIII. DATA STRUCTURE SHORTHAND

It is frequently desirable to display pictorially the relationship between records. This is particularly important in developing an overall view when planning an information system. A special graphic technique has been developed to display records and their master-detail (chain) relationships.

This technique uses a block shape to designate a record type and an arrow connecting two blocks to designate a chain. The arrow points from the master to the detail. This picture of block, arrow, block carries the following message: 1) there are some number of records in the system of the master type; 2) each of these records is the master of a chain of the specified type; 3) there are some number of records of the detail type (0, 1, 2, 3, . . ., n) in

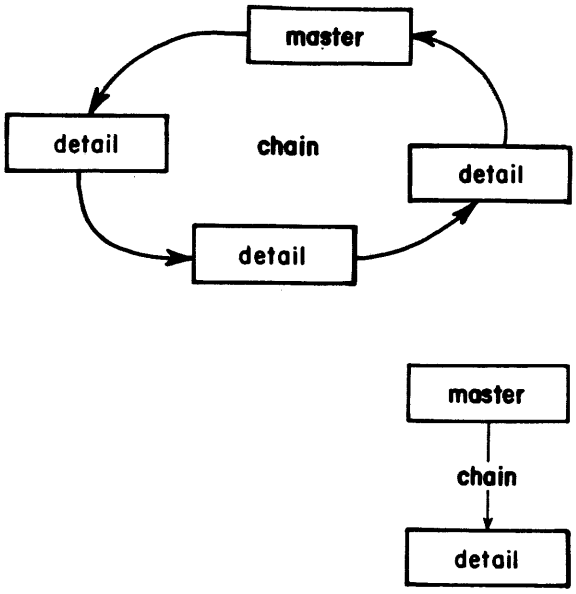

Figure 8. IDS Data Structure Shorthand.

each such chain. Using this graphic technique, very complex data structures may be presented in a condensed and understandable form.

It is believed that the long sought information algebra may be developed around this notation. A new set theory is needed in which the master record is represented by the empty set, and details represent the ordered members of the set.

X. SAMPLE DATA STRUCTURE (PURCHASE ORDER)

The information contained on a purchase order can furnish an example of how information might be structured.

Looking at a purchase order form, three groups of information may be seen. One group is concerned with information about the vendor, i.e. his name, address, and vendor code. Another group is concerned with information about the order, i.e. the order number, due date, mode of transportation, and dollar value. The third group is concerned with the information about a particular item to be purchased, i.e. its identification, description, quantity, unit price, and extended dollar value. Three different records might be designed in order to carry the information contained in these three groups. These three records would be a vendor record, order record, and item record. If a purchasing information system were established along these lines, there would be a vendor record for every vendor with whom the business is concerned. The vendor record would be the master record of an order chain. There would be an order record for each order currently stored in the system. It would be a detail in an order chain. Each order record would, in turn, be the master of an item chain. This item chain would contain one or more item records depending on the number of items on the purchase order. This example contains three records and two chains. The vendor record is only a master. The order record is both a master and a detail. Finally, the item record is only a detail. The IDS Data Structure Shorthand shows all this with only three blocks and two arrows. Very complex systems with thirty or more record types have been clearly described using the IDS shorthand.

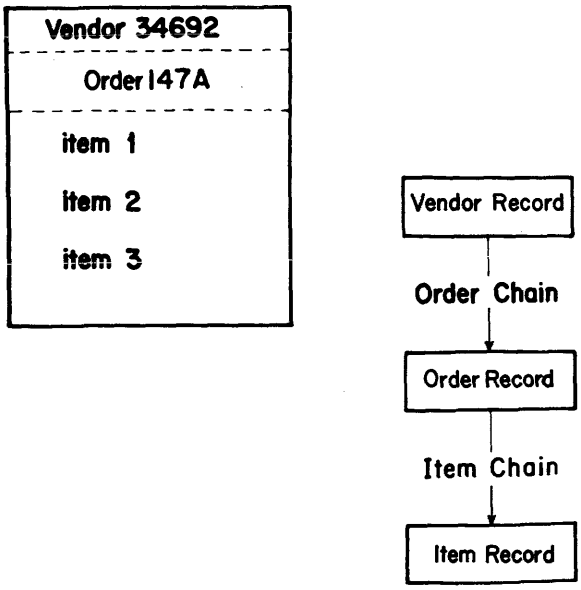

Figure 9. Purchase Order Data Structure.

A data description for the sample problem is shown in figure 10. Each record must be clearly defined as to the data fields which it contains as well as the chains in which it participates. The appropriate IDS controls for storage and chaining must also be described.

A look at part of a network created by the vendor, order, and item records illustrates both the need for the data structure shorthand and

Data Type	Record Name	Chain Name	Field Name	IDS Controls	Field Image
Record	VENDOR			Calculated	
Field	VENDOR		VENDORNO	Unique	X(6)
Field	VENDOR		VENDORNAME		X(18)
Field	VENDOR		ADDRESS		X(24)
Field	VENDOR		CITY STATE		X(24)
etc. for all fields in VENDOR record					
Chain Master	VENDOR	ORDERCHAIN		Sequenced	
Record	ORDER			Calculated	
Field	ORDER		VENDORNO	Redundant	
Field	ORDER		ORDERNO	Unique	X(12)
Field	ORDER		ORDERDATE		999
etc. for all fields in ORDER record					
Chain Detail	ORDER	ORDERCHAIN			
Chain Control	ORDER	ORDERCHAIN	VENDORNO	Match	
Chain Control	ORDER	ORDERCHAIN	ORDERNO	Ascending	
Chain Master	ORDER	ITEMCHAIN		Sequenced	
Record	ITEM			Secondary	
Field	ITEM		ORDERNO	Unique, Redundant	
Field	ITEM		ITEMNO	Unique	X(3)
Field	ITEM		MATLIDENT		X(18)
Field	ITEM		ORDERQTY		9999V9
etc. for all fields in ITEM record					
Chain Detail	ITEM	ITEMCHAIN		Prime	
Chain Control	ITEM	ITEMCHAIN	ORDERNO	Match	
Chain Control	ITEM	ITEMCHAIN	ITEMNO	Ascending	

Figure 10. Purchase Order Data Description.

its power. The arrows indicate that one record "points" to the next record in a chain and that each chain "closes" onto the master record of the chain. Regardless of how many chains a record is in, the record exists in the file only once. It may be "pointed at" by many other records.

Still another way of illustrating the data contained in the sample problem is shown in figure 12. Here the chaining is represented by the appropriate reference addresses.

X. *Procedural Commands*

The functional verbs PUT, GET, MODIFY, and DELETE, previously introduced, require further explanation for better understanding. These verbs may be used in a GECOM, COBOL, or FORTRAN sense. In fact, there is reason to wonder when they will move out of the developmental world and become part of the industry standard languages. Perhaps, a better phraseology would be to ask whether the industry languages committees will take advantage of IDS to catch up with data storage units, real time, and command and control systems programming. They are part of the COBOL compiler language for the GE–400 and 600 series computers now.

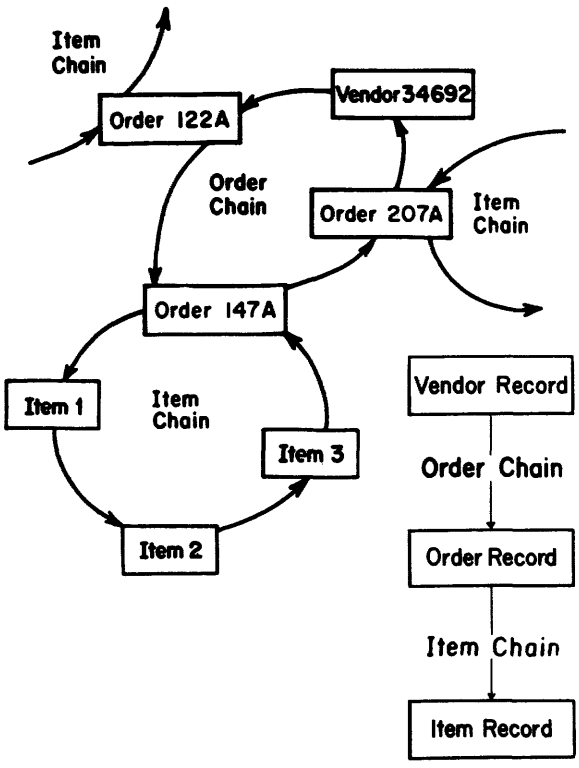

Figure 11. Purchase Order Data Structure.

VENDOR RECORD

REF ADDRESS	VENDOR NO	VENDOR NAME		ORDER CHAIN NEXT	
100	34692	ABC CO.	ETC	322	ETC

ORDER RECORD

REF ADDRESS	ORDER NO		ORDER CHAIN NEXT	ITEM CHAIN NEXT
285	207A	ETC	100	287

ORDER RECORD

REF ADDRESS	ORDER NO		ORDER CHAIN NEXT	ITEM CHAIN NEXT
322	147A	ETC	285	335

ITEM RECORD

REF ADDRESS	ITEM NO	MATERIAL IDENT	QTY	ITEM CHAIN NEXT
335	1	75L38	10	337

ITEM RECORD

REF ADDRESS	ITEM NO	MATERIAL IDENT	QTY	ITEM CHAIN NEXT
337	2	122A93	310	342

ITEM RECORD

REF ADDRESS	ITEM NO	MATERIAL IDENT	QTY	ITEM CHAIN NEXT
342	3	46A95PI	2	322

Figure 12. Purchase Order Data Structure.

The following are examples of procedural statements which would cause the IDS to execute certain actions. The letters in capitals are the required words of the IDS language. The lower case letters are data and procedural variables, i.e. record names, chain names, field names, and sentence names.

A. *Example 1—PUT.* "PUT vendor RECORD." This command stores a new vendor record in accordance with its data description. Its fields' values would be picked up from working storage and packed into the new record skeleton in a data block. The order chain field in the new record would be packed with the reference code of the new vendor record itself because there are no details at this moment and the next record in the order chain is the vendor record.

B. *Example 2—GET.* "GET NEXT order RECORD OF order chain, OR IF vendor RECORD GO TO location~a." This command would retrieve the next record of the order chain and unpack its fields into working stor-

age. If the next record is an order record, the control would be transferred to the succeeding command. If the next record is a vendor record, control would be transferred to the command identified by the sentence name "location~a." The actual record retrieved is not accessible to the programmer—however, the contents of its data fields are unpacked and made available in working storage. This serves two purposes. First, protection is given to the data in the file in a father-son type sense. Second, it means that the data from a record will remain in working storage until another record of the same type is retrieved and unpacked into the same working storage fields. For example, if an item record were first retrieved, followed by its order record, then the order record's vendor record, the fields from all three records would be simultaneously available in working storage for processing.

C. *Example 3—MODIFY.* "MODIFY CURRENT item RECORD, REPLACE quantity FIELD." This command will modify the current item record, i.e. the last item record accessed, regardless of what has transpired since it was processed. It will pack the content of the working storage field "quantity" into the corresponding data field of the record replacing the existing value. A modify command will modify one or several fields in accordance with those specified in the command. Fields may also be modified by adding or subtracting the contents of working storage to that of a record. The appropriate commands would be MODIFY recordname RECORD, ADD fieldname FIELD, or SUBTRACT fieldname FIELD.

It was mentioned earlier that fields were frequently used to sequence the detail records in a chain. These fields are called sequence control fields. If a sequence control field is modified, the detail will be automatically delinked from its master and relinked to it again in accordance with the new value of its sequence control field.

Fields are also used to control the selection of the master records and chains in which to insert detail records. These fields are called match control fields. If a match control field of a detail record is modified, the record will automatically be delinked from its old master. Its new master will be retrieved, and the record linked to its new master according to the ordering rule specified for the chain.

D. *Example 4—DELETE.* "DELETE vendor RECORD, IF ERROR GO TO error~a." This command will retrieve the vendor record specified by the code stored in the "vendor code" field in working storage. If there is no vendor record with that specific vendor code, the command will respond by setting up an error code to specify the nature of the "fault" and transfer control to the command identified by sentence named, "error~a." If the vendor record is successfully retrieved, its deletion process will begin. If a vendor record is to be deleted, its order chain must be deleted too. Consequently, if a vendor record is to be deleted, its order chain must be searched to ascertain that there are no order records in it. If there are order records, they must be deleted before the vendor record is deleted. In the same manner, an order record may not be deleted if there are any item records in its item chain. Therefore, all item records in an item chain must be deleted before the order record is deleted. This makes the deletion command a very powerful command and one to be used with due respect.

Two optional features have been provided which aid the programmer in using the delete command. If the programmer anticipates that order records may be linked to the vendor record (that a detail may be linked to its master), he may wish to print a control report of the orders deleted by using the phrase "AND IF order RECORD PERFORM reportline~1" with his delete command. This will cause the deletion process to be interrupted everytime an order record has been deleted. The subroutine identified by the sentence name, "reportline~1" will be executed. Because of the frequent desire to produce some form of a control report on deletions, the delete command actually retrieves and unpacks, into working storage, the data fields of a record prior to deleting it.

The second optional feature of the DELETE command permits the programmer to attach an escape phrase. For example, the programmer might attach the phrase, "BUT IF order

RECORD GO TO statement~a." In this case, the detection of an order record as a detail to the vendor record would immediately terminate the deletion command without the order record, its item records, or the vendor record being deleted. Using the DELETE command in this manner, the programmer need not test to determine the presence of an order record prior to initiating the DELETE command. He may boldly set out to delete a master record, and still escape from the deletion if there is a detail that he wants to protect.

All of the commands, GET, PUT, MODIFY, and DELETE, permit the addition of error test and branch. The user is urged to use them so that he is immediately aware of any fault that occurs and the nature of that fault. A typical fault that could occur during a "PUT" is attempting to put a duplicate record which is prohibited according to the data definitions.

XI. RECORD RETRIEVAL SPECIFIERS AND RULES

Three of the IDS macro instructions require that a record be retrieved so that it may be operated on. GET means retrieve a record and unpack its data fields into working storage. MODIFY means retrieve a record and modify specified fields in the record according to the command the the contents of working storage. DELETE means retrieve a record, unpack its data fields into working storage, delete any detail records, and finally, delete the specified record. Only the PUT command lacks the retrieval aspect. It means find space for a new record, link it into its chains, and pack its data fields from working storage.

There are six different retrieval rules from which the programmer may choose. These rules may be used in conjunction with the functional processes; GET, MODIFY, and DELETE. Examples 2, 3, and 4 in Section X used three of these rules, respectively, "NEXT OF CHAIN," "CURRENT" and the associative retrieval rule which is specified by the absence of a record specifier adjective. These rules may be sub-divided into two classes. The two rules which are absolute in their nature, i.e. there is only one record that satisfies their specifications regardless of when they are executed. They will be discussed first. The other four rules are relevant to what has transpired previous to their execution.

A. "_____" *specifier.* The absence of a specifier indicates that the record to be processed is identified by the data values stored in the fields of working storage. The particular fields concerned are those fields which have been described in the data description of the specified record as the unique fields for that record.

B. *"DIRECT" specifier.* This specifies that the record is to be retrieved based on the reference code (address) stored in the communications field named, "QDIRF" (*DI*RECT *REF*ERENCE). The programmer may store any reference code there and then retrieve the record associated with that reference code.

The IDS system is so designed that once the reference code is assigned to a record, it is permanent. The addition or subtraction of data storage units will not affect it. The modification of the record to add or delete either data or chain fields or modify their content will not affect it. In fact, the uniqueness and permanence of the reference codes make them ideal candidates for dual use as reference code and invoice number, order number, pay number, vendor code, customer code, drawing number, or stock number.

C. *"CURRENT" specifier.* The "CURRENT" record specifier instructs the system to reretrieve the last record of that type processed by a GET, PUT, or MODIFY command. If the last command executed for a given record type were a DELETE command, the last record would have been deleted and it would be impossible to retrieve the current record of that type because there is none. This would create a "fault" and an error would be signalled.

D. *"NEXT" specifier.* The "NEXT" specifier is one of a set of three chain processing specifiers. These specifiers require that a chain name be appended so that the specification would be complete. As an example, the command below specifies that the programmer wishes the program to access the next item record in the item chain:

"GET NEXT item RECORD OF item CHAIN." The particular record accessed

clearly depends upon which record is the current record in the item chain when the command is executed. An IDS command must have been executed prior to executing any of the chain processing commands. This prior command must have accessed a record in the desired chain and therefore established the current record in the chain. Only then does the phrase "NEXT RECORD OF CHAIN" have any meaning.

E. *"PRIOR" specifier.* The "PRIOR" specifier is used to specify that the chain is to be processed in a backward direction. This command contains the same restraint as the "NEXT" specifier, that the current record of the chain must have been established. The ability to process a chain backwards is optional and dependent upon the chain having been specified in the data description, as a "PRIOR" chain.

F. *"MASTER" specifier.* The "MASTER" specifier directs the chain processing to proceed directly to the master record of the specified chains, accessing but ignoring all intermediate detail records. The optional specification of the chain as a "HEADED" chain provides an additional pointer field in each detail record containing the reference code of the master record. In the presence of this option, the "MASTER" specifier will proceed directly to the master record without accessing the intermediate detail records. As with the other chain processing specifiers, the chain must have been accessed and a current record established, prior to executing a "MASTER" command.

Alternate Retrieval. The retrieval rules using the record specifiers, NEXT, PRIOR, and DIRECT exist under conditions where the type of record to be retrieved cannot always be predicted. These commands, therefore, permit the insertion of one or more "OR IF recordname RECORD GO TO sentence name" phrases. The program logic is then able to branch to the specified sentence following the execution of the functional portion of the command in accordance with the record retrieved. The use of an "IF recordname RECORD GO TO sentence name" phrase (note the "OR" is missing) will cause the program logic to branch after retrieval, but *before* the execution of the functional portion of the command. This permits the execution of the functional portion of the command (GET, MODIFY, or DELETE) when using the record specifiers DIRECT, NEXT or PRIOR on selected record types and the by-passing of the function if other types are retrieved. As an example, the following command might be used:

"DELETE NEXT order RECORD OF order CHAIN, IF vendor RECORD GO TO sentence~a."

The repeated use of this command would delete successive order records which are in the order chain. However, when the vendor record is retrieved, it will not be deleted and control will be transferred to sentence~a.

The data structuring abilities of the IDS permit the definition of more than one detail record type in a chain. In the case of the PRIOR and NEXT OF CHAIN retrieval actions, unspecified record types in the chain will be accessed and skipped over until a record of a specified type is retrieved. If the chain is completely traversed without the retrieval of a specified record type, an error is signalled. The retrieval of an unspecified record type by the DIRECT specifier will cause an error to be signalled.

Error Conditions. All of the commands of the IDS are structured with the provision for an error statement. As an example:

"GET item RECORD, IF ERROR GO TO sentence~b."

If this command were attempted and had failed because no item record could be retrieved with an order number and line number, matching those in working storage, then the program control would be transferred to sentence~b. This permits the program to test whether the function has been carried out successfully. If the command has not been successful, the error condition may be tested to determine the nature of the fault and the appropriate action initiated.

XII. HISTORY OF DEVELOPMENT

Historically, the IDS's foundation in well disciplined data structure goes back to the file

structures developed by General Electric at Hanford for their 702 Report Generator and File Maintenance System.[2] These structures reached greater generality and power in the SHARE 9PAC system which was largely guided and programmed by GE Hanford and supported by The Dow Chemical Company, Union Carbide Company, GE Heavy Military Electronics Department and others. The deliberations of the SHARE committee on The Theory of Information Handling[3] also contributed to the early thinking on the Integrated Data Store.

The current implementation of the Integrated Data Store is based on a set of free standing subroutines written in the General Assembly Program language for the GE–200 series computers. The original version was prepared as an adjunct to GECOM. It had a compiler generator which processed data definition cards and IDS macro instructions and produced mixed GECOM and General Assembly Program statements which were subsequently compiled by GECOM to produce an object program. The macro instructions were executed as generated in-line coding. IDS was first operated in this form in January, 1963, with hand compiled macro instructions. The application that it was applied to was the IDS compiler-generator itself which was used to generate IDS coding for subsequent application programs. The first completely generated program was a product materials file maintenance and explosion routine. This routine was used during the summer of 1963 to run comparative speed tests with another routine performing the identical tasks which had been hand coded employing conventional disc programming techniques. The machine generated IDS program ran twice as fast as the comparison program and used less file space to store the data.

During the Fall of 1963, the current implementation of the IDS was programmed. This version switched from compiled in-line coding to an interpretive subroutine organization with calling sequences. This is another step in the separation of the data structure from procedural logic and parallels the dictionary used by 9PAC which was brought together with the procedure at load time. The parametric version appears to operate at about the same speed as the in-line coding version.

XIII. SUMMARY

The Integrated Data Store is an operational tool for programming the GE–225 with Disc Storage Unit. It automatically processes the complex file maintenance and retrieval problems presented by a data storage unit. It gives a high degree of file protection and through data structuring and redundancy elimination, it accomplishes considerable file compression. The user has the option of many storage and retrieval techniques. It yields efficient programs with buffered operation of the disc file. The requirement to structure the data before programming greatly reduces redesign and debugging problems. IDS provides for the first time an effective method for describing the complex interrelationships of data present in most information systems. It further provides the means for efficiently processing and maintaining these in the environment of a mass memory system. It moves list processing techniques out of current limitations of core memory and thus makes them available for practical data processing.

We challenge the national standards committees for COBOL, FORTRAN, and ALGOL and the designers of the "New Programming Language" to survey their current accomplishments, which are many, and to determine whether the above capabilities offered by IDS should be added to their languages.

REFERENCES

1. KAVANAGH, T. F., "TABSOL—A Fundamental concept for Systems Oriented Languages," Eastern Joint Computer Conference, December, 1960.
2. MCGEE, W. C., "Generalization—Key to Successful Electronic Data Processing," Journal ACM, January, 1959.
3. SHARE Committee on Theory of Information Handling, "TIH #1 Report," 1959.
4. BACHMAN, C. W., "The Integrated Data Store Function Specifications," General Electric Co. internal publication, January, 1962.