FORMAT-INDEPENDENCE AND CONTENT-ADDRESSING IN DATA BASES

Acknowledgments

The work described in this thesis was carried out between October 1967 and December 1970 while the author was at Imperial College.

I wish to thank the many organisations and individuals who contributed directly or indirectly to this work. My sincere thanks are due to G.F. Coulouris for devoting much time to general discussion and for providing constructive criticism and encouragement at all stages of my work. I wish to express my gratitude to ICL for sponsoring this Ph.D work and for facilities and technical assistance generously provided. I would particularly like to thank R.W. Mitchell (ICL R.A.D.O.) for advice on hardware during my design work for the CAS system. My thanks are due to Queen Mary College for providing computing facilities during the implementation of the CRH system. I would also like to express my gratitude to all the organisations involved in the survey work of this thesis and particularly to the individuals who readily gave their time to our discussions.

Abstract

This thesis describes two experimental Data Management systems developed using a common integrated approach to system design. An integrated approach involves consideration of three things during the development of a new system:- user requirements, existing software and hardware. Only by consideration of these three things is it possible to ensure the relevence of system design and development. In addition to providing a context for system design the integrated approach attempts to circumvent the undesirable dichotomy of computing into separate disciplines of "hardware" and "software". The bias and presentation of the work in thesis is towards the latter, primarily because the author's background has been mainly in this area. However, the ideas behind the work derive from considerations of user, software and hardware.

The first system to be described is a Content-Addressable Store (CAS) system concerned primarily with inter-record relations. The user requirement identified was content-addressing. Existing systems provide this facility inefficiently on current direct-access hardware. Existing hardware for content-addressing was found to be unsuitable for meeting the user requirements identified. The CAS system is based on the design of a new class of hardware device (CAS) for content-addressing and described at a functional level in this thesis together with its supporting software.

The second system to be described is a Commercial Record Handling (CRH) system concerned primarily with intra-record relations. The user requirement identified was format-independence. Existing software systems (at the time of designing the system) did not offer this facility. Hardware suitable for the implementation existed in the form of the Basic Language Machine. The CRH system is a software implementation of a simple Data Management system which provides format-independence as its main feature.

The CAS and CRH systems are complementary, since the former is concerned with inter-record relations and the latter with intra-record relations.

Contents

Chapter 1 : Introduction

Chapter 2 : Literature and User Surveys

2.1 General

2.2 Classification of Data Management Systems

2.3 Data-Structures and Storage-Structure Independence

2.4 Data Independence

2.5 Real-Time User Survey

2.6 Batch-Processing User Survey

2.7 Summary of Literature and User Surveys

Chapter 3 : A Content Addressable Store (CAS)

3.1 General

3.2 Design of a Content Addressable Store (CAS)

3.3 The ECAL Language

3.4 Performance Evaluation

3.6 Further Considerations

Chapter 4 : A Commercial Record Handling (CRH) System

4.1 General

4.2 The CRH Language Facilities

4.3 Implementation of the CRH System

Chapter 5 : Summary and Conclusions

5.1 CAS System Summary

5.2 CRH System Summary

5.3 Conclusions

Bibliography

Appendix A : Real-Time User Reports

Appendix B : Batch-Processing User Reports

Appendix C : Detailed ECAL Language Description

Appendix D : Simulation Program Outline

Appendix E : Detailed CRH Language Description

Appendix F : Listing of Program Example

Chapter 1 : Introduction

Computerised Data Management Systems are a class of software systems which aim to provide their users with convenient facilities for storing and manipulating large quantities[1] of related information. A broad spectrum of these systems exist with differing characteristics[2]. Data Base Management Systems (DBMS) which represent the most sophisticated end of this spectrum are characterised by the fact that the collection of related information (data base) is recognised as an entity in its own right. The data base has its own description which is available to DBMS software and which exists independently of user processing programs. A distinction is made between the abstract relations which a user may declare (data-structure) and the implementation of those relations within the computer system (storage-structure). All Data Base Management systems attempt to insulate their users from storage-structure (storage-structure independence) with varying degrees of success. Within this context there are still many variations in the nature and generality of Data Base systems. All provide at least minimal batch processing facilities and a great many provide real-time capabilities.

The existence of a separate data base description can confer on programs an important degree of flexibility called "Data-Independence". This term implies a measure of insulation between a compiled program and the data with which it interacts. Depending on the degree of data-independence various changes to a data base description can be accommodated without recompiling the program. This facility has significant effects on the flexibility of an information processing system, allowing progressive evolution of data base and the addition of application programs with minimal perturbation to existing applications. Data-independence is

---

1 Typically orders of magnitude greater than the capacity of c.p.u. main memory.

2 A classification of Data Management Systems is described later.

achieved by delaying the "binding" of program and data till after compilation, unlike many existing languages for data-processing. Binding is frequently either a pre-run activity or an execution time activity where the program "interprets" the data base description. The Codasyl DBTG proposals [1] provide data-independence by their "schema" and "sub-schema" data description facilities. Data independence is an insulation between two logical descriptions of data whereas storage-structure independence is an insulation between logical facilities for describing and manipulating data, and their physical implementation.

Major differences between Data Base Management systems arise in the nature of the data-structure relationships they provide. There are many different frameworks for discussing data structure relationships, one which is suitable for the purposes of this thesis is from the point of view of inter and intra-record relations. The term "record" suffers from a variety of interpretations so it is essential to define its usage within the context of this thesis. First it is important to distinguish between classes of object and the class members since data descriptions are primarily class descriptions, the members of which constitute the data base. An item is a description of a class whose members are fields. Fields hold data values and are the atomic units of which the data base is composed. Generally a data base is subdivided into disjoint collections of highly related fields these collections are called records. Records invariably exhibit regularities of structure and separate into different classes. We will term the description of a record class a record schema, record schemas contain items describing the fields of member records. It is unclear whether the existence of records results from the need for a conceptual grouping of data in problem analysis or for the convenience of

---

1 The DBTG proposals are referenced and discussed in Chapter 2.

the Data Management System. Data Management systems frequently take advantage of the existence of records by storing their fields contiguously (or nearly so) whereas the individual records may be widely dispersed. A result of this (or maybe a reason for it) is that relations are divided into two classes intra-record and inter-record relations, the former being supported much more effeciently on current hardware.

Two important inter-record relations are explicit relations and associative relations and in general Data Base Management systems select one of these ways as the primary method of expressing inter-record relations. Explicit relations are named relationships between records which can be declared in the data structure and which are treated differently from the data within the records (for example the "sets" of the DBTG proposals). Explicit relations are usually supported by direct access "pointers" in the storage-structure linking related records. Further, in many systems store management economics curtails the generality of these pointer structures to tree-like forms instead of more general networks. Even if network relations are supported the availability of predominantly explicit relations can limit the usefulness of an existing Data Base for new applications of that data. Associative relations allow records to be related by commonality of content and imply content-addressing facilities. Systems which provide predominantly associative relations usually have a language form called a selection criterion (described in 3.1) which allows records to be accessed by specifying any part or parts of their content. Content-addressing is supported on current hardware by complex storage-structures frequently involving heavy use of "indexing" techniques. The main advantage of content-addressing facilities is in flexibility since the use of a Data Base is not constrained by a relatively sparse set of explicitly declared relations. The intra-record relations (i.e. between items in the same record schema) tend to be rather simpler than the inter-record relations. Generally some grouping of items is permitted giving rise to tree-like structures. If all records in a class are exactly similar, and comprise fixed length

fields, then the record class is termed "fixed". If variations are permitted from record to record (i.e. fields with different lengths or complete fields omitted) then the record class is termed "variable". Data Management systems vary considerably in the degree of variability (if any) which can occur, restricted variability simplifying the storage structure of records.

Earlier the term "data-independence" was introduced as a desirable facility, allowing changes to be made to the data base without necessarily invalidating compiled processing programs. One important class of changes is where intra-record structures (described by record schemas) are changed, insulation of user program from this class of change we will term format-independence. Record schema changes occur when new items needs to be inserted in the record schema for example, or when the breakdown of the data base into records is altered by merging record classes to create a new class. Intra-record structure is naturally affected by this type of change and a format-independent system will allow a variety of changes to intra-record structure without invalidating user programs whose data item requirements are still met by the altered record schemas. Explicit inter-record structure can also be heavily affected by changes such as merging record classes, associative inter-record relations are far less affected.

The Content Addressable Store (CAS) system and the Commercial Record Handling (CRH) system described in this thesis are complementary, being concerned with inter and intra-record relations respectively. The CAS system provides efficient hardware evaluation of selection criteria against a data base, providing performance advantages over existing systems. The CRH system provides format-independence for fixed record structures as part of its design goals, since a sufficient number of relatively simple applications exist where fixed record structures are suitable and the special architecture of the Basic Language Machine offers significant advantage in this case. The provision of a full Data Base Management system by integrating the CRH and CAS systems is certainly viable but possibly restricted by the fixed intra-record structure, although implementation of certain

existing Data Base Management Systems are similar in this respect. Another point to be considered is that requirements for intra and inter-record relations are not independent - the provision of powerful inter-record relations can alter requirements for intra-record structure. This thesis does not, however, propose that the two systems be integrated but that they represent investigations into two important and related areas of Data Base Management.

Chapter 2 : Literature and User Surveys

2.1 General

2.2 Classification of Data Management Systems

2.3 Data-Structures and Storage-Structure Independence

2.4 Data Independence

2.5 Real-Time User Survey

2.6 Batch-Processing User Survey

2.7 Summary of Literature and User Surveys

Section 2.1 General

This chapter contains two classes of survey work, a literature survey to examine existing Data Management systems and techniques, and user surveys to examine user requirements relating specifically to the two systems described later in the thesis.

The literature survey is in three sections. The first section is a classification of Data Management systems on the basis of three important aspects of user-orientation data-structures, storage-structure independence and data independence. "Data-structure" is manifested in relationships between items (intra-record structure) or between records (inter-record structure). Three basic classes of relation are identified - sequence, explicit and associative relations. Sequence relations imply some data ordering and sequential access, explicit relations imply graph structures with data addressing paths along interconnections, associative relations are established by commonality of data content. Storage-structure independence and data-independence were discussed in the Introduction, they represent "insulation of the data-structure from its implementation" and"insulation of a program data-structure from the data base data-structure" respectively. Format-independence is the term used in this thesis for data-independence only for intra-record structure. The second and third sections (2.3 and 2.4) of the literature survey discuss the three aspects of user-orientation in more detail.

The user surveys appear in two separate sections. The Real-Time User survey was conducted to provide information for the Content-Addressable Store (CAS) system described in chapter 3. The Batch Processing User survey was conducted for user information relevant to the Commercial Record Handling (CRH) system described in chapter 4.

A summary of the results of the survey work is provided in the final section of this chapter.

Section 2.2 Classification of Data Management Systems

At the lowest end of the classification are the simple batch-processing File Systems comprising a collection of 'input/output" functions which serve to conveniently transfer unstructured records between user program and file,dealing with buffering and blocking of records on secondary storage. The data-structure facilities consist only of inter-record relations, the user having minimal facilities either for sequential or direct addressing on a single index. File systems are provided by computer manufacturers as part of their basic software e.g. ICL Standard Housekeeping[1] and IBM Basic Access Methods (BSAM, QSAM and BDAM) [2]. The input/output functions are embedded in the machine assembler which itself allows the records to be manipulated within core-store. More powerful File Systems are necessary to provide equivalent facilities in a multi-access environment. Each time-sharing operating system is closely linked to a suitable File System - George III has its own File System [3], the Cambridge Titan File System [12] and the MULTICS File System [5] are other examples. These File Systems, together with the operating system provide more advanced facilities such as sharing of files and privacy arrangement, security and file-recovery in event of machine malfunction or user error. However they provide very little independence from storage-structure, and no data-independence. These features are usually provided by higher levels of Data Management Software implemented above the file system.

The Data Processing Languages COBOL [6], BCL [7], PLI [8] and NEBULA [9] provide more user oriented facilities than direct use of file system and assembler. High level facilities for describing record classes are provided offering tree-like intra-record structures varying from the fixed COBOL structures to the completely variable NEBULA structures. In this context "variable" means that record fields may vary in length from member to member in a record class and/or may optionally occur. The high-level record structure

facilities provides storage-structure independence for intra-record structure (i.e. for data in core-store). The inter-record structure is largely the same as for the File Systems providing little insulation from storage-structure. The Data Processing Languages do not provide data-independence, however COBOL provides more flexibility in this direction than the other languages by its separation of procedure and data declarations. The main aim of such languages is to provide machine-independent programs, a goal which can be achieved provided suitable compilers are available and the different machine File Systems have broadly similar capabilities.

Information Retrieval Systems are a class of special purpose systems which provide comprehensive and rapid retrieval facilities on a (frequently) large data base. Generally retrieval is in real-time but update and maintainance are in batch mode. A remarkable feature of this class of Data Management systems is the diversity of purpose-built systems which exist, most large computing facilities having one or more of their own tailor-made systems. Recent surveys in the UK[10] and the USA [11] list a variety of such systems currently in use. The bulk of Information Retrieval Systems provide for document retrieval, prime examples being Library Information Retrieval Systems [4]. The System Development Corporation offer a general purpose time-shared document retrieval system called ORBIT [13],.ICL offer a system called PEARL [14]. Information Retrieval Systems provide highly user-oriented facilities only for the retrieval function and therefore are not classed as Data-Base Management Systems. Nevertheless, they provide storage-structure independence and an associative inter-record structure only matched in generality by the most sophisticated of the Data-Base Management Systems (see later). Information retrieval systems generally have a simple intra-record structure and the retrieval language is generally interpretive so that questions of data-independence are not relevant. Although a wide variety of storage-structures and search-optimisation algorithms are used, very many of the systems

use variations on the "inverted" and "multilist" storage-structures relevant to Data Base Management systems and described in some detail later.

Data Base Management Systems provide a full range of user-oriented data management facilities (database creation, retrieval, update and frequently report generation facilities) but vary considerably in their detailed characteristics. Generally they provide storage-structure independence and varying degrees of data-independence. The CODASYL systems Committee have published a survey[15] of Data Base Management systems which compares a variety of systems on the basis of selected "features". A clear distinction is made between storage-structure and data-structure, however data-independence was not discussed. This is surprising since the CODASYL DBTG proposals [16] explicitly include data-independence by their schema/sub-schema facilities. Data Base Management systems fall into two main classes, the procedural systems oriented toward batch-processing and the non-procedural systems which provide interactive facilities and batch. Olle[24] equates user-orientation with non-procedurality and on this basis the non-procedural systems are the most highly user-oriented of all Data Management systems.

The procedural Data Base Management systems PLUTO [18] IDS[19] IMS [20] and NIMMS [21] are "Host Language" systems - they rely on a procedural host language for computational facilities. With a host language system the transfer of records between the data base and central processor is explicitly under user control. The host language, usually COBOL and/or machine assembler, is used to process the records in main memory. The DBTG proposals upgrade the COBOL language to the status of a procedural Data Base Management system. The procedural systems fall into groups with respect to inter-record structure, NIMMS and IMS having a variable tree-like structure and IDS, PLUTO and DBTG COBOL having network structures. All five systems provide degrees of storage-structure independence. IDS and DBTG COBOL are alone in allowing the user

to optimise certain aspects of storage-structure without affecting the logical facilities of the data-structure:- both allow degrees of control over relative placement but DBTG COBOL provides more far reaching control by its "pointer array" or "chain" alternatives for "set" representation. High degrees of data-independence are offered by NIMMS, IMS and DBTG COBOL though IDS and PLUTO are less able in this respect.

The non-procedural Data Base Management Systems TDMS [22] GIS[23] and UL1 [24] provide a complete set of user facilities without needing a host language. They may be used interactively via a terminal or in batch mode, the latter frequently being used for large volume updates and general data-base maintainance. The three systems have basically the same variable tree-like data structure though the language facilities differ. In addition associative inter-record relations can be used. Complete insulation from storage-structure is provided. The implementations of the systems appear mainly interpretive so that data independence is also ensured. An important feature of these systems is that the user does not have to control the transfer of data between the main memory and data base. Making use of the associative facilities a user defines the particular operation he wishes to perform and a selection criterion for a particular file. The selection criterion defines the subset of file records to which the operation must be applied by the Data-Base Management System, but does not prescribe the method by which this is to be carried out.

Section 2.3 Data-Structures and Storage-Structure Independence

A Data Management System maps the user data-structure into a particular storage-structure which is itself dependent on the hardware capabilities. It is likely that the most efficient use of the hardware will occur if the data-structure is close to the hardware capabilities, also the storage-structures are likely to be much simpler in this case. However, the provision of user-oriented facilities tends away from the rather limited capabilities of existing hardware. Clearly there is room for compromise, the Data Base Management systems tending towards the user rather than machine.

Virtually all Data Management Systems provide Sequence relations which occur primarily in inter-record structure. Sequence relations provide a striking example of the undesirable effects of storage-structure dependence. In a commercial computer installation with magnetic tape (serial) secondary storage a large proportion of available computer time is spent re-ordering (sorting) the files. Since the user sees the sequence in which file records are stored the storage-structure needs to be changed if the user requires a different sequencing. The result is a large proportion of unproductive work simply altering storage-structure. However, this situation is unavoidable if the secondary storage is serial. A conclusion which may be drawn is that sequencing is generally not an intrinsic property of the data, rather it is a requirement of a particular application using the data. It is desirable that the user should be independent of sequence relations in the storage-structure but essential that he should be able to declare sequence relations in his data-structure. The responsibility lies with the data management system to provide a general sequencing capability within the data-structure which is independent of the storage-structure. This can only be provided using direct access storage.

Both associative and explicit data-structure relations are in evidence in the Data Base Management Systems. These relations can be storage-structure independent but their generality differs considerably between systems. The storage-structures required for supporting the two relations are quite different. For example TDMS system which provides both relations handles them separately as described in two papers by Bleir[25] and Bleir and Vorhaus[26] - The former describing the explicit relations and latter the associative relations.

Explicit relations may occur either as inter or intra-record structure. In the case of the latter they invariably occur as the tree-like record structures provided in many Data Management systems and present few implementation problems. Explicit inter-record relations are present in the Data Base Management systems and vary from the tree-like structures of NIMMS and IMS to the networks of DBTG COBOL, IDS and PLUTO. The explicit relations seen in the data-structure frequently correspond to direct access pointers in the storage structure particularly the case of PLUTO and to a lesser extent IDS. The logical appearance of the data-structure is quite strongly dependent on the physical storage-structure. DBTG COBOL provides full storage-structure independence since the logical operations on the network (sets) are affected only in performance by alterations to storage-structure. The storage structures for supporting networks are more complex than for tree structures and the management of the former can prove problematic (PLUTO requires twelve separate program "runs" for file reorganisation). Lefkovitz [27] describes two different storage structures for supporting networks, these correspond with the "chain" and "pointer" array options indicated in DBTG COBOL.

Associative relations occur only as inter record structure and by their nature provide storage-structure independence, records being related only by commonality of content (content-addressing). Simple content-addressing (which because of its simplicity is usually not classed as such) is where records have an

index. The user may retrieve an individual record by providing a value for one of the record fields nominated as an index. Index field values are different for each record and the Data Management system maintains an index of value-address pairs which is searched to give record addresses. This is a particularly simple storage-structure and as a result all Data Management Systems including File Systems provide such a facility. Full content-addressing facilities are present only in the information retrieval systems, non-procedural Data Base Management systems and certain specialised systems mentioned later. Full content-addressing is made available by a construct called a selection criterion (see 3.1) which allows a very large number of possible relationships to be established rather than being confined to a fixed relatively sparse set of explicit relations. The storage-structures for supporting associative relations are very complex and inefficient on current hardware and hence are present only in the more sophisticated systems. Lefkovitz [27] describes the two basic storage-structures "multilist" and "inversion" (described also in 3.1) which can be used to support the general associative relations. The two structures have different trade-offs in terms of search and update time but are not necessarily mutually exclusive - an inverted file system could be considered as a multi-thread system where each thread length is unity. Lefkovitz advocates a "controlled length" multi-thread system where the maximum thread length is a parameter supplied when the files are created or reorganised thus allowing optimisation (without affecting the data-structure facilities). Hsiao and Harary [28] have formalised a description of storage-structures which includes the inverted and multilist structures as special cases. A "Generalised Retrieval Algorithm" for selection criterion retrieval is described in terms of the formalism and hence is logically equally applicable to both the inverted and multilist storage-structures. The systems TDMS, GIS and ULI all use inversion but Information Retrieval Systems use either structure.

The "Relational Model" [29] for data bases proposed by Codd is accessed by a Data Sublanguage [30] based exclusively on content-addressing. The constructs available for this purpose are somewhat more complex than in either the Information Retrieval or Data Base Management Systems mentioned earlier. A selection criteria refers to a particular class of records but expressions in the Data Sublangauge permit the formation of expressions involving many classes by use of "range variables" and "quantifiers" together with multiple "qualifiers". Each qualifier is equivalent to a selection criterion and the evaluation of a Data Sub-language expression can be performed by multiple selection criterion operations together with the storing and processing of intermediate results.

In addition to the Data Base Management systems there are certain special purpose languages designed to provide associative relations for particular applications. Feldman's LEAP [31] language uses "hash coding" to support the associative relations which take the form of "triples" rather than selection criteria. LEAP is used primarily for artificial intelligence applications rather than data processing. Graphics applications, which require complex relational data structures, have stimulated the design of a variety of "ring-processors". Gray[32] has conducted a broad survey of these systems which augment machine assembler to provide general associative relations via a "ring storage-structure" which is partially visible to the user. The ring processors only allow their data to exist within core-store but LEAP uses a large volume of direct access secondary storage. FIND[33] is a package offered by ICL for providing content-addressing in batch mode largely for use in report generation. The package performs an exhaustive search of files to evaluate selection criteria.

Section 2.4 Data Independence

Data-Independence relates to the "binding" of program and data. In the Data Processing languages the binding is performed at compile-time and changes to the data descriptions always necessitate recompilation. The effects of changes to inter-record structure are largely undefined but for intra-record structure the compiler is able to check the validity of program references to items and reject invalid program statements for subsequent modification. COBOL mitigates the "freezing" effects of compile-time binding to some extent by its separation of data and procedure divisions. Systems which accommodate data changes entirely through recompilation are not generally regarded as offering data-independence.

The procedural Data Base Management systems differ in their approaches to binding and hence in their data-independence facilities. Systems like IDS have compile-time binding for intra-record structure (no format-independence), but inter-record structure is bound at execution time i.e. tables representing inter record relations are accessed at run-time. NIMMS and IMS which provide a high degree of data independence exclusively use tables for inter and intra-record structure. These systems allow a program to be compiled to some sub-description, then the tables are produced in a pre-run activity linking program sub-description to the data base description. The DBTG COBOL proposals include in broad terms a description of the type of Data Management System envisaged for supporting the languages and it would appear from this that the object versions of the "schema" and "sub-schemas" are equivalent to the tables of NIMMS and IMS giving comparable degrees of data-independence.

The non-procedural Data Base Management systems do not have the concept of a program sub-description and a linkage phase since they are implemented interpretively using the database description directly.

The provision of data-independence poses problems not apparent in the preceding discussion. Is it possible to establish rules for permissable changes to data base description ensuring that a program which was written for the original database is still valid?

The generality or profusion of facilities offered by the system affects this question. In NIMMS and IMS which restrict the inter record relations to tree-like forms it is feasible to generate suitable rules. Although from first hand user experience of one of these systems this author believes that a consistent set of rules which is not too restrictive is very difficult to establish. The problems increase when inter-record relations become more complex. Two current approaches are evident at the present time, the essentially pragmetic approach of the DBTG proposals or the systematic attempt at formalising data description and manipulation of Codd's "Relational Model" based on predicate calculus. In the latter the data base is seen as a number of n-ary relations[1] which can be accessed through a "data sub-language α" which provides exclusively content-addressing. Codd argues that "explicit relations" inhibit the ability to achieve data-independence, his model provides a set of formal operations (projection, selection, join) for altering the relations with predictable effects on the expressions of the data sub-language.

---

1 In this context "relation" is used in the mathematical sense.

Section 2.5 Real-Time User Survey

2.5.1 Overview

2.5.2 Questionnaire Structure

2.5.3 Summarised Answers to Questionnaire

2.5.1 Overview

A survey of large real-time users was carried out in the initial design stages for the CAS to obtain performance and design goals and to obtain a deeper understanding of the general data handling requirements of large scale systems. This survey was conducted in the USA during the spring of 1970. Contact with users was made beforehand via letter and enclosing a questionnaire (see 2.5.2). Also included in the survey were some individuals and organisations whose work relate to the CAS system. The organisations visited were:-

Bell Telephones, Holmdel Laboratories, New Jersey.
The future Data Base requirements of the Bell Telephone Company were discussed and an existing small scale data base system was described.

Boeing, Seattle, Washington.
Existing Data Base systems connected with aircraft manufacture were discussed.

California Department of Motor Vehicles, Sacramento, California.
The extremely large real time system (AMIS) operated by the department was discussed in detail.

Chicago University, Chicago, Michigan.
A library Information Retrieval system for Chicago University was discussed.

Illinois University, Champaigne, Illinois.
L. Parker was engaged in a project involving hardware for content-addressing.

NASA, Washington DC
A document information retrieval system was discussed in detail.

System Development Corporation (SDC), Los Angeles, California.
SDC produce and market the TDMS Data Base Management System which was the object of our visit.

Stanford University, Menloe Park, California.
Professor J.A. Feldman (the designer of the associative language LEAP) discussed content-addressing generally. A library information retrieval system (SPIRES) for the university was examined in detail.

The structure of the questionnaire is described in 2.5.2 and this is followed in 2.5.3 by summarised answers to the questions. A detailed report on each of the users appears in Appendix A.

2.5.2 Questionnaire Structure

The questionnaire contained a total of twenty questions which split into the following categories:-

(i) Extent of Real Time Committment.

(ii) Use of Generalised Software.

(iii) Desirable Hardware Features.

(iv) Updating, Security and Recovery.

(v) Privacy.

(vi) Maintenance.

(vii) Operation System Features

(viii)Quantitative Questions.

The actual questions and summarised answers under the above headings can be found in the next sub-section.

Category (i) above was aimed at establishing the overall magnitudes of the data bases, the relative amount of work performed in real-time and the desirability of doing more if the transaction processing capacity were increased. Category (ii) examined the extent to which generalised software had been used for system implementation. Category (iii) was an attempt to establish whether increased hardware assistance for data management had been considered. Category (iv) included some main problem areas of data sharing in real-time systems. Category (v) questioned privacy arrangements for the multiple users of the real-time systems. Category (vi) questioned whether the real-time system must be put "off the air" for data base maintenance. Category (vii) comprised some questions on operation systems, particular whether "time slicing" was employed for a guaranteed response. Category (viii) included number of quantitative questions, for example data base size and response times. Also included was a question on the record structures of real-time data bases.

2.5.3 Summarised Answers to Questionnaire

For each of the categories described in 2.5.2 the individual questions are listed followed by summarised answers from all the users.

Extent of Real Time Commitment:-

Q1 What is the total size (in bits) of your information store? How many different data bases are there?

Q2 What proportion of your data bases are used for real-time applications? What are the maximum, minimum and average sizes of these data bases?

Q3 Would some of your batch processing jobs be done in real time if your transaction-oriented system capacity were increased?

The total size of the information store varied from 7×10^{10} to 5×10^{8} bits. In all cases except the Chicago library system and Boeing the whole of this information was available for real-time processing, though not necessarily for a whole working day. Generally, more work would be done in real-time if the transaction processing capacity could be increased. However, the point was made that there will always be a requirement for batch processing, mainly for report generation and statistical analyses.

Use of Generalised Software:-

Q4 What kind of software has been provided for your transaction-oriented system by

a. Computer manufacturers?

b. Independent Software Suppliers?

Are there any general conclusions to be drawn about the adequacy (or otherwise) of the above software?

Q5 What proportion of your transaction-oriented system software has been written by yourselves?

Generalised Data Base Management Software had not been used to implement any of the operational systems. This probably reflects the fact that only recently have suitable Data Base Management systems become available. Bell Telephones who had not yet implemented their Data Base system planned to use manufacturers' software exclusively. Although, some doubts were expressed about the adequacy of this software for their complex real-time update requirements (see later). All of the operational systems had been implemented by user programmers employing some standard manufacturers packages for terminal handling and other low level activities.

Desirable Hardware Features:-

Q6 Are there any hardware features which you think would be helpful in implementing transaction-oriented systems?

Faster access time for direct access storage was generally considered desirable.

Updating, Security and Recovery:-

Q7 What type of updating procedure(s) is (are) adopted:

(a) amendments to files are batched and the updating done overnight.

(b) records are updated in place.

(c) new records are produced in real-time and a system of historic records kept on:

(i) other storage media

(ii) the same storage media

Q8 Are two or more users allowed to operate on the same data base and if so what security measures are provided for the following situations:

(a) two or more processes attempt to read the same record

(b) two or more processes attempt to update the same record at the same time

(c) a process attempts to read or update a record which is in the process of being updated

(d) when a process has a complicated update involving many records can the file be locked out from all other processes?

Q9 What type of security is employed to provide recovery facilities for:

(a) the system

(b) the individual user who wishes to recreate his files?

All users except Boeing and Bell use batch updating procedures. In the batch update situation there are no contention problems in record access. Security involves taking copies of files en bloc and copying transactions so that files may be restored to different points in time. Bell, Boeing and the projected version of SPIRES all require real-time update facilities. The TDMS system provides real-time update but requires the complete file to be locked out during the update process, this falls short of user requirements noted in this survey. The question of update locks for records is viewed apprehensively by Bell and for the projected SPIRES. The latter require record lockout for update but they have no satisfactory proposals as to the form of this at present. Bell have doubts whether the computer manufacturers' Generalised Data Base Management software can meet their exacting requirements. Namely, that records must be locked during update, although the locked version must be available for reading only. Furthermore, they require a convenient facility for being able to 'undo' a particular transaction restoring the previous state of a record some time (a matter of a day or so) after an update. Boeing, on the other hand ignore the problems of 'clashes' during real-time update since they consider the possibility remote. Recovery requirements also differ, at present Boeing have no recovery facilities but later this year plan to start taking security copies of their real-time files. Both Bell and projected

SPIRES require some form of transaction logging capability in addition to periodic file dumping.

Privacy:-

Q10 What type of password system is employed to allow access to data bases and is the collection of Data Bases hierarchical from this point of view.

Name of the systems employed a 'password' system for privacy checking. In most cases the terminals themselves were recognised by their 'address', this was sufficient to prevent unauthorised access to certain data. The Chicago Library system will require some form, as yet undecided, of password system in the future.

Maintenance:-

Q11 Is Data Base maintenance able to operate in parallel with terminal activity (e.g. index maintenance and file reorganisation)? Is this a desirable feature?

The answers varied considerably, some systems able to perform maintenance in parallel with terminal activity whereas in others the file being reorganised was inaccessible. In SPIRES maintenance occurred in parallel, the 'update' function performing a clean-up of the disk store. The NASA system makes files unavailable during reorganisation, which occurs very infrequently.

Operating System Features:-

Q12 Is time-sharing dependent on program events or are processes timesliced?

Q13 What facilities are available for inter-process (or inter-terminal) communication and synchronisation:

(a) shared flags

(b) shared communication file

(c) other?

Q14 What facilities are available for changing the storage media of real-time files?

In all cases time-sharing was dependent on program events. The message switching facilities of the operating system were used, no additional inter-process communication was required, inter-terminal communication was generally allowed but rarely used. Apart from the security 'dumping', real-time files always existed on a direct access medium, no paging to a slower medium occurred as with multi-level file stores.

Quantitative Questions:-

Q15 What are the maximum, minimum and average numbers of records in your transaction-oriented data bases?

Q16 What is the size of your records (maximum, minimum, average) and what type of record structures are used?

Q17 Have any statistics been compiled on the distribution of activity at the terminals and record hit-rate?

Q18 What response time is considered acceptable?

Q19 How many terminals can the system support with an acceptable response time?

Q20 What are typical sizes (core occupancy and c.p.u. time required) for each transaction?

None of the user's had compiled the statistics of question 17 above. For question 15 average answers only were supplied. For question 20 transaction sizes were not provided. Record structures, when discussed, were variable with variable length fields and optional occurrences.

Answers to the above questions appear in Fig. 2.1.

		No. of Records	Record Sizes (bytes)	Response Time	No. of Terminals	c.p.u. Time per Transactor
Chicago Library		30,000	250 avg.	1 min.	single	not known
SPIRES	present	9,000	300-600	10 sec.	small numbers	not supplied
	projected	50,000	300-600	10 sec.	25	not supplied
AMIS	present	19M	59-474 130 avg.	2.5 sec.	200	500 ms approx.
	projected	14M	61-4064 190 avg.	2.5 sec.	200	500 ms approx.
Boeing		1M	100-1024	30 sec.	50	100 ms
Bell		1M	100-3000	5 sec.	400	not known
NASA		650,000	100-2000 800 avg.	20 sec.	60	several seconds *

* The large c.p.u. time arises from index searching and key correlations (partially inverted file structure).

Fig. 2.1 Answers to Quantitative Questions

Section 2.6 Batch Processing User Survey

2.6.1 Overview

2.6.2 Questionnaire Structure

2.6.3 Summarised Answers to Questionnaire

2.6.1 Overview

A survey of four selected batch-processing users was carried out in order to validate the basic facilities already defined for the CRH system, particularly the suitability of the intra-record structure and if possible gauge the desirability of format-independence. This survey was conducted in Britain during the summer of 1968. Contact with the users was made via the ICL Sales Organisation, all four users having ICL equipment and were engaged in commercial Batch processing work. The four users were:-

i. The Automobile Association (the AA), London.
The AA is a very large motoring association.
Data Processing work includes the maintainence of a membership file and car insurance files.

ii. BP (Llandarcy), South Wales
British Petroleum have an oil refinery at Llandarcy.
Data Processing work includes Invoicing, PERT, Stock Control, Ledgers, Statistics and Payroll.

iii. Dunlop Semtex, South Wales
Dunlop Semtex is part of the consumer group of the main Dunlop organisation, they manufacture and supply floor covering. Data Processing includes large customer file maintenance, Invoicing, Ledgers and Statistics.

iv. GKN (South Wales)
GKN (South Wales) are part of the main GKN organisation and manufacture reinforced bars, nails, wire and similar products. Their Data Processing work includes Invoicing Ledgers, Stock Recording, Payroll and Statistics.

Detailed reports for each user appear in Appendix B.

2.6.2 Questionnaire Structure

The questionnaire contained a total of twelve questions which split into the following categories:-

(i) Programming Language used.

(ii) I/O or c.p.u. bound.

(iii) Record structure.

(iv) Files with multiple record types.

(v) Files used by more than one program.

(vi) System changes.

(vii) Program testing.

The actual questions and summarised answers under the above headings can be found in the next sub-section.

Categories (i) and (ii) were aimed at establishing whether the user had a definite requirement for a purely high level language and the complexity of the computational part of his Data Processing. Category (iii) established the sort of record structures used, to validate the intra-record structure capability of the CRH system. Category (iv) was concerned with the processing of multiple record types on the same file, particularly whether different record types needed to co-exist in the c.p.u. during processing. Category (v) examines the degree to which files were used by more than one program, the greater the degree the more beneficial format-independence is likely to be. Category (vi) explicitly asked about the effect of system changes to establish whether format-independence could have reduced the side-effects of these. Category (vii) gave an indication of the amount of system changes which occurred over a period of time.

2.6.3 Summarised Answers to Questionnaire

For each of the categories described in 2.6.2 the individual questions are listed followed by summarised answers from all the users.

Programming Language:-

Q1 What programming languages are used?

Three of the four users programmed exclusively in assembler,

the remaining one used COBOL because of time pressures on program implementation. Assembler was preferred, mainly on grounds of machine efficiency.

I/O or c.p.u. Bound:-

Q2 In the normal running of the installation in what ratio are the computer times separated?

(a) dealing with file transfers

(b) In-core processing.

In all four cases the programs were I/O limited.

Record Structure:-

Q3 Files on cards or paper tape:

(a) Are variable length records used, how often?

(b) Is the variability -

(i) Simple repetitions of a given element?

(ii) Variable length alphanumeric?

(c) Are these files used as direct input into a program or are they first put to tape/disk?

(d) Are these files prepared sometimes from documents external to the system, e.g. customers order forms?

Q4 Are variable length records used on magnetic tape or disk? Could the variable parts be confined to the last fields only?

Q5 Is the conventional COBOL tree structure capable of representing all the file record structures?

For slow input media (cards/paper tape) the record structures were mainly fixed for cards and variable with character terminated fields for paper tape. In all cases the slow input was batched and validated to disk or tape, transformed as necessary. The slow input was invariably punched from internal documents allowing the DP manager control over document format. The records on disk or tape are largely fixed, where variable information occurred it

could be repositioned at the end of the record without difficulty. The COBOL record structure was deemed adequate by all four users.

Files with multiple record types:-

Q6 How many files have more than one record type on them? What is the maximum number of different record types held on the file?

Q7 When files which contain multiple record types are being read, is it necessary to have more than one record existing in core (or can each incoming record, overwrite the previous record)?

The percentage of files with multiple record types varied between 10 and 60% for the different users. Usually there were about five different record types for these files, although, in one exceptional case there were fifty-six different types. (This was a cumulative "movements file" for a number of different master files). In general different records from the same file do not need to co-exist in the c.p.u. (i.e. record areas overlap).

Files used by more than one program:-

Q8 Are there files which are used by more than one program? In all cases a significant proportion of files were used by multiple programs, usually between one and five programs. For two users 90% of their files were used by more than a single program.

System Changes:-

Q9 How often does the system change, and in what ratio are the changes reflected in the two factors below?

(a) Program changes

(b) File format changes.

Q10 What system changes have occurred which lead to re-programming?

Q11 How much extra space is allowed in records for additional information if the system changes?

Apart from the User (the AA) proceeding with system integration the systems were considered relatively static. However, for three of the users it was relatively easy for the people concerned to recall a system change which lead to re-programming.

The changes were either an insertion of new field in the record or extending an existing field. In one case, recalled by the AA, fourteen programs were altered because a field was extended, three of these were altered because they accessed the field and eleven were altered because of the new format alone. The one user (GKN) who did not appear to have this type of problem used indirect addressing to access record fields (see Appendix B.3). The indirect addresses could be altered to accommodate file format changes.

Program Testing:-

Q12 How much time is taken up testing new programs (and alterations to existing programs).

Apart from the user who was still implementing his system, the time spent program testing varied from a small percentage up to 20% for the AA who were integrating their systems.

Section 2.7 Summary of Literature and User Surveys

Literature Survey Work

Full content-addressing facilities (i.e. selection criteria) were provided by the Information Retrieval systems and the non-procedural Data Base Management systems. All other systems provided simple content-addressing in the form of indexing facilities (either single indexes or multiple). In the case of the procedural Data Base Management systems the indexing facilities were accompanied by explicit inter-record relations taking the form of "trees" or "networks". However, Codd's Relational Model" is based entirely on content-addressing facilities and has no explicit inter-record relations. The fact that indexing was provided by all systems but only the more complex systems provided full content-addressing derives largely from the complexity and inefficiency of the storage-structures for supporting the latter on current hardware. (These structures are examined in greater detail in 3.1).

Data-independence was provided by the Data Base Management systems but not by the Data Processing Languages. Questions of Data-Independence impinge primarily on the procedural Data Base Management systems because the non-procedural systems are generally interpretive in nature. Certain of the non-procedural systems provided data-independence only for inter-record structure while others provided in addition data-independence for intra-record structure (format-independence), the DBTG proposals fall into the latter category.

User Survey Work

The Real-Time User Survey produced the following list of requirements for a Data Base Management System suitable for the users interviewed:-

(i) Data Base Size:
Useful Data Base sizes are in the range 10^8 - 10^{11} bits.

(ii) Record Sizes:
Average record sizes are in the range 100 - 800 bytes but individual records may be considerably larger.

(iii) Transaction Handling Capacity:
The system should be capable of supporting two to four hundred terminals, however up to fifty terminals would be acceptable in many cases.

(iv) Content-Addressing Response Times:
Full content-addressing facilities were desirable with response times in the region 1-5 sec although in certain cases up to 30 sec could be tolerated.

(v) Sequential Access:
Batch processing facilities must be provided, implying that efficient methods of processing data sequentially must be available.

(vi) Record Locking:
Update in real-time is required, certainly with the ability to lockout individual records.

(vii) Record Structure:
Record structures should include variable and optional fields.

(viii) Recovery:
File recovery facilities should be provided.

(ix) Privacy:
Privacy facilities will eventually be required (though they are not used at present).

The Batch-Processing User Survey indicated that for the users interviewed:-

(i) The intra record structure of the CRH System was suitable for the applications considered.

(ii) The facilities for storing different record classes on the same file appear suitable.

(iii) The format-independence facility would have reduced the re-programming effort for specific system changes for each user.

It is always dangerous to extrapolate to the total population the results of small surveys, particularly as the sample was chosen on the basis of availability rather than strictly random. However, it is felt that the systems discussed were representative of significant bands of computer applications and the results obtained endorse the relevance of the two research areas (content-addressing and format-independence) to be described later in this thesis.

Chapter 3 : A Content Addressable Store (CAS)

3.1 General

3.2 Design of a Content Addressable Store (CAS)

3.3 The ECAL Language

3.4 Performance Evaluation

3.5 Further Considerations

Section 3.1 General

In this chapter a Content-Addressable Store (CAS) system is described. This system comprises special purpose hardware and supporting software which together provide content-addressing capabilities. The terms "content-addressing" and "associative store" bring to mind a host of devices and proposals. Hanlan [34] in a survey lists the characteristics of a variety of such devices. The device proposed in this thesis differs from these in three significant ways:-

(i) The capacity of the content-addressable store in a CAS (i.e.~10^8 bits) is a factor of one hundred times larger than the greatest of these other devices.

(ii) The time taken to perform a content-addressing operation is many orders of magnitude larger in the CAS system (typically a CAS might take 1 second whereas content-addressing in many of the other devices is comparable in speed with a ferite core store).

(iii) The CAS performs a number of content-addressing operations in parallel.

Content-addressing is the accessing of data through a partial specification of its content. Full content-addressing capabilities are made available to a user by a construct called a selection criterion. As an example assume the existence of a class of personnel records, one for each person, which include items "AGE", "POSITION", "SALARY", "FORENAME", "SURNAME", and "MANNO" - a selection criterion might be:-

AGE < 25 AND (SALARY > 3000 OR POSITION = "MANAGER")

items: AGE, SALARY, POSITION

values: 25, 3000, "MANAGER"

comparison operators: >, = ,<

boolean operators: AND, OR

The important characteristics are:- that it consists of a number of triples each triple comprising a comparison pair and a comparison operator (e.g. AGE, 25 are a comparison pair in the triple AGE < 25). The triples are formed into a boolean expression involving "AND", "OR" and bracketed as desired. The selection criterion identifies a subset of the personnel record class, those for which the selection criterion is true. There are variations on this basic form of the selection criterion.

Cross-coupling is where a comparison pair consists of two items instead of item and value. For example in a record class used to record stock levels in a production plant two record items might be "STOCKLEVEL" and "REORDERLEVEL" , the former indicating the current quantity and the latter the minimum level before a new order to replenish is issued. A selection criterion involving cross-coupling which defines the subset of stock requiring replenishment is:-

STOCKLEVEL < REORDERLEVEL

User requirements for content-addressing facilities were evident in the user survey of 2.5 and from the existence of such facilities in the Information Retrieval systems and more sophisticated of the Data Base Management systems. Existing software techniques for supporting content-addressing and some relevant hardware devices are discussed next before going on to outline the hardware-aided approach which forms the basis of this chapter of the thesis.

Storage-Structures for Content-Addressing

An obvious method of supporting content-addressing for a collection of records is by an exhaustive search, examining each record individually and evaluating the selection criterion to produce the value "true" in which case the record is required for processing or false when it is discarded. Unfortunately, if the collection of records is large (occupying upwards of 10^8 bits) this process

is usually grossly uneconomic because each record must be transferred to the c.p.u main storage for software evaluation of the selection criterion. Channel capacity is generally being wasted since only a small proportion of the actual data transferred is required. Furthermore, even if this wastage can be tolerated the time taken to perform the search, assuming current data transfer rates and cpu bit-matching rates, precludes real-time evaluation of selection criteria. Current implementations of content-addressing generally employ one of two divergent storage structures to minimise the data which has to be processed to locate the "hit records" (i.e. those for which the selection criterion is "true").

The Multilist storage-structure illustrated in Fig. 3.1 contains a separate list for each item which may appear in a selection criterion, the list has direct access pointers threading through all records containing the item in question and a directory contains symbolic identifiers and the heads of each list. The directory is relatively simple in structure and small in comparison with the data. The Inverted storage structure illustrated in Fig. 3.2 has a much larger directory consisting of an index for each item. The index is a vector of value-pointer pairs and corresponds to a single thread of the multilist structure but is stored separately from the records. A fully inverted structure is where there is an index for each record item, a partially inverted structure is where only a subset of the record items are indexed. The total storage requirements of both structures is similar if item values stored in index entries do not appear within the records. Certain fully inverted systems like PEARL [14] and the Bell Labs system (see Appendix A) dispense with records entirely, item values occur only in the directory where "record number" is used to associate item values deriving from the same record and this can (at great expense) be used to recreate records. Other systems like GIS duplicate item values between the records and the indexes. The TDMS system employs a fully inverted storage structure with extensions in order to accommodate the terms dependent on intra record structure, a table (CFIND) containing a representation of

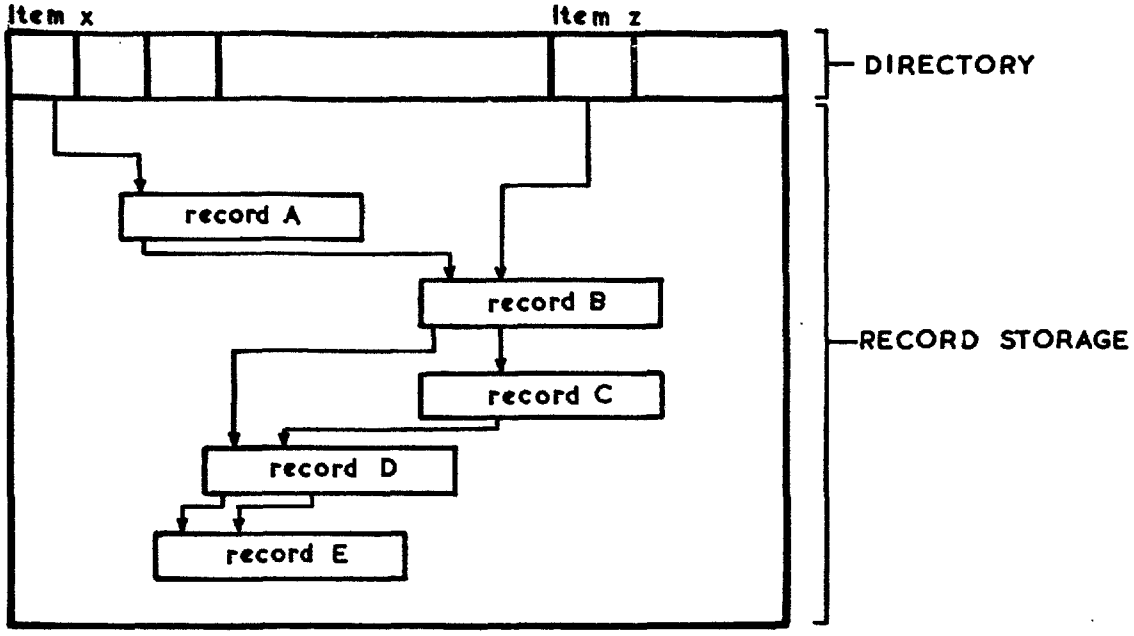


Fig. 3.1 Multilist Storage-Structure

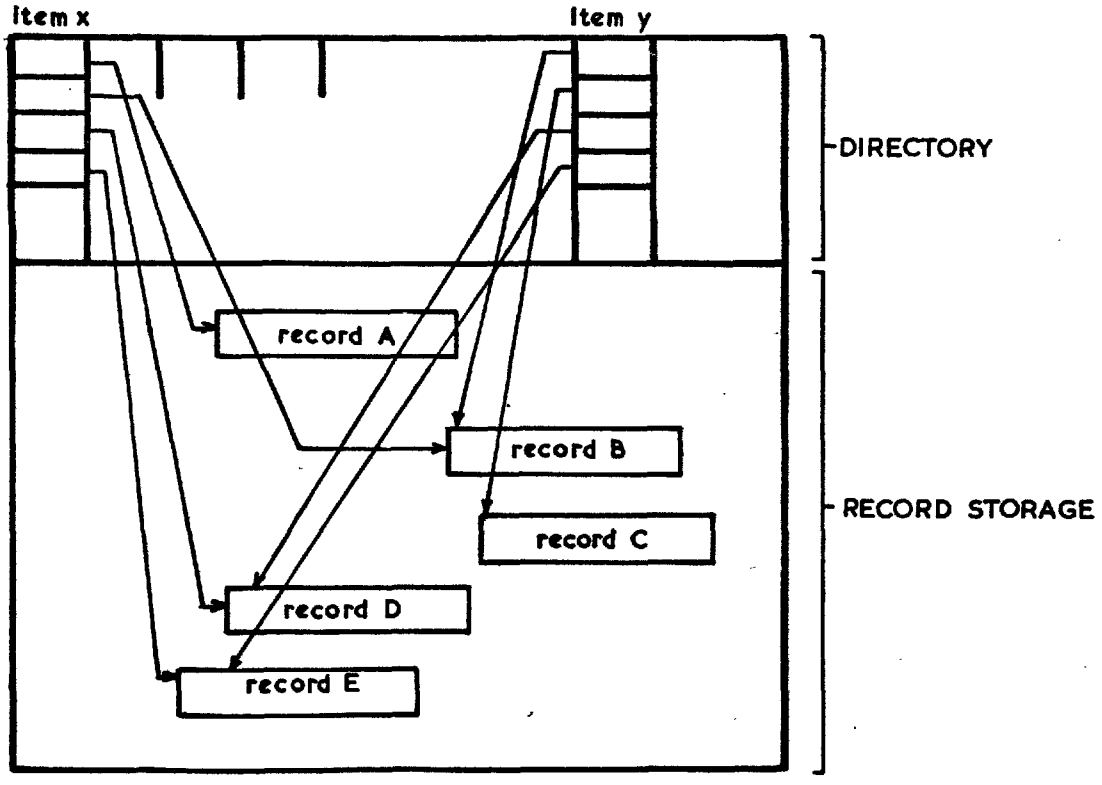


Fig. 3.2 Inverted Storage-Structure

the explicit tree-like intra record structure is interposed between index and record storage. A variation on inversion where instead of a separate index for each item an index whose entries comprise an ordered collection of items is proposed by V.Y. Lum [35] . In this case the retrieval response is more rapid than a conventional inverted structure but storage and update overheads are larger. Information retrieval systems frequently use the multilist structure because of its implementation simplicity. The SPIRES system (see Appendix A) is an example. Lefkovitz provides a comprehensive treatment of these structures and a comparison for design purposes. As a rule of thumb the inverted structure provides faster evaluation of selection criteria but a slower update capability. Both structures require complex system software (particularly the inverted structure) and result in either heavy I/O channel usage (multilist) or heavy cpu and main store overheads (inverted).

Relevant Existing Devices

Apart from the "associative stores" referred to in the introduction to this chapter there are a number of devices which provide degrees of content-addressing based on rotating magnetic storage (as is the CAS). The IBM 2314 Magnetic Disk [36] provides a hardware assisted content-addressing mechanism, designed primarily to aid the implementation of index sequential file organisations. The performance of the device when used in content addressing mode viz. the low rate of data scanning (10^6 Bits/sec), limited forms of the comparison hardware and the lack of parallel evaluation of search criteria, combine to make the device unsuitable for the design targets of the CAS system. Many small magnetic drums have a very limited content-addressing capability, again as an attempt to speed up indexing, associations take the form of fixed format word pairs where the content of one word can be used to retrieve the corresponding word. These facilities suffer from similar defects to the IBM 2314 and additionally the content-addressable store has very low capacity.

Two devices have been proposed with broadly similar aims to the CAFS, these are the Singer Data Processing System [37] and the Illinois Logic-Per-Track Disk [38] . The former is described in a Patent and has certain operational similarities to CAS system. It is a step towards a hardware solution for content-addressing but suffers from certain disadvantages when compared with the CAS . The main disadvantages of the Singer system are:-

(a) Fixed length records

(b) Lack of parallel evaluation of selection criteria

(c) Primitive comparison logic

(d) Wastage of hardware utilisation where the selection criterion only specifies part of a record (as will almost always be the case)

Parker, describes a method of utilising a head/per track disk for information retrieval. Basically this is the result of alternative design philosophy to that adopted for the CAS and will be considered in more detail in 3.2.1.

Hardware-aided Content-Addressing

Software techniques for providing content-addressing have been discussed earlier, they have disadvantages in performance and software complexity. The alternative approach proposed in this thesis is the construction of a special purpose hardware Content-Addressable Store (CAS) which acts as a peripheral unit to a conventional cpu as in Fig. 3.3. Data is stored in the CAS as a collection of unordered variable records. The CAS controller is capable of autonomously executing different types of task, the most important of which is the search task. A search tasks performs a content-addressing operation, retrieving information from those records which satisfy a selection criterion. The search task is performed by storing a representation of the selection criterion termed a selector in the CAS controller which exhaustively scans

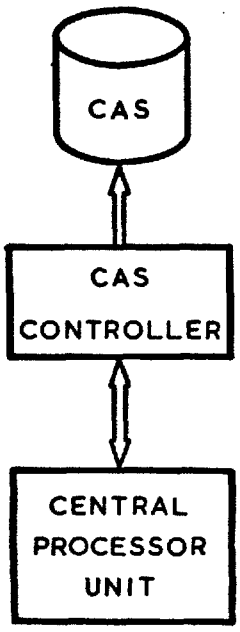


Fig. 3.3 A Content-Addressable Store (CAS)

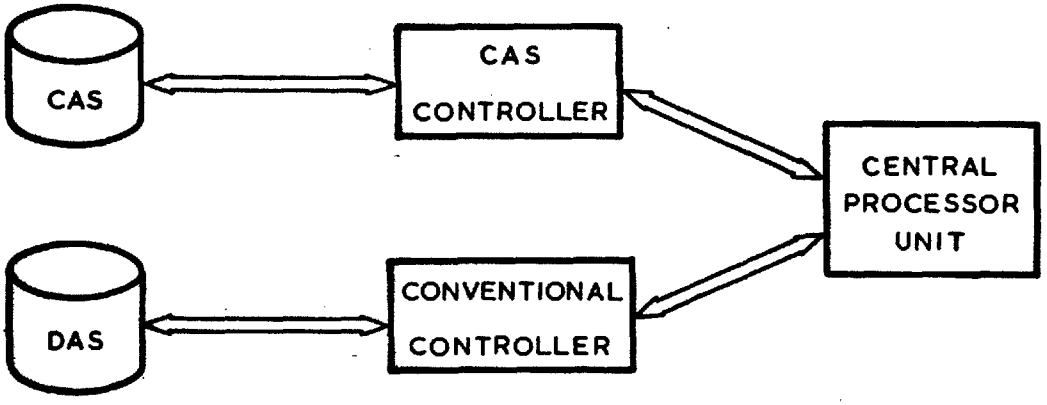


Fig. 3.4 A Hybrid System

the records in the store evaluating the selection criterion for each record in turn. Records which satisfy the selector (hit records) are not transmitted bodily to the cpu, instead only those fields specified by a retriever (associated with the selector) are transferred. A search task specification issued by the cpu to the CAS controller therefore comprises a selector and a retriever, and the effect of the specified task is to transmit subsets of hit record fields to the cpu. Selectors may include any of the record fields so that access to CAS records is equivalent in generality to a fully inverted system based on a conventional direct access storage device. The CAS controller is able to execute to completion a number of search tasks simultaneously during a single exhaustive scan of the store. Halatsis [17] has worked from the device description and "Controller Command" specifications of this chapter to produce a suitable hardware design (down to gate level) for a CAS verifying that realistic figures for capacity and scan time are 10^8 bits and 1 sec. based on current technology.

Many alternative configurations are possible containing a CAS and conventional peripheral devices. One attractive possibility is indicated in Fig. 3.4 where the CAS is used in conjunction with a conventional direct-access storage of larger capacity. The CAS is used to provide content-addressing for a larger data base stored in the DAS, this hybrid system corresponds in logically to a partially inverted system. The methods of utilising such a hybrid system are outlined in Section 3.5 and also in two papers published separately [39] , [40].

The major part of this chapter is concerned with the CAS and associated software,assuming for convenience that no other storage devices are involved. Fig. 3.5 indicates the main software elements which would be present in an operational system. User programs and users at terminals see a high-level language for expressing search tasks. This is converted by the Data Base Management software into a task specification understood by the device handler. The device handler then converts this into the task specification understood by the CAS controller. This latter transformation is extremely simple involving minor alterations.

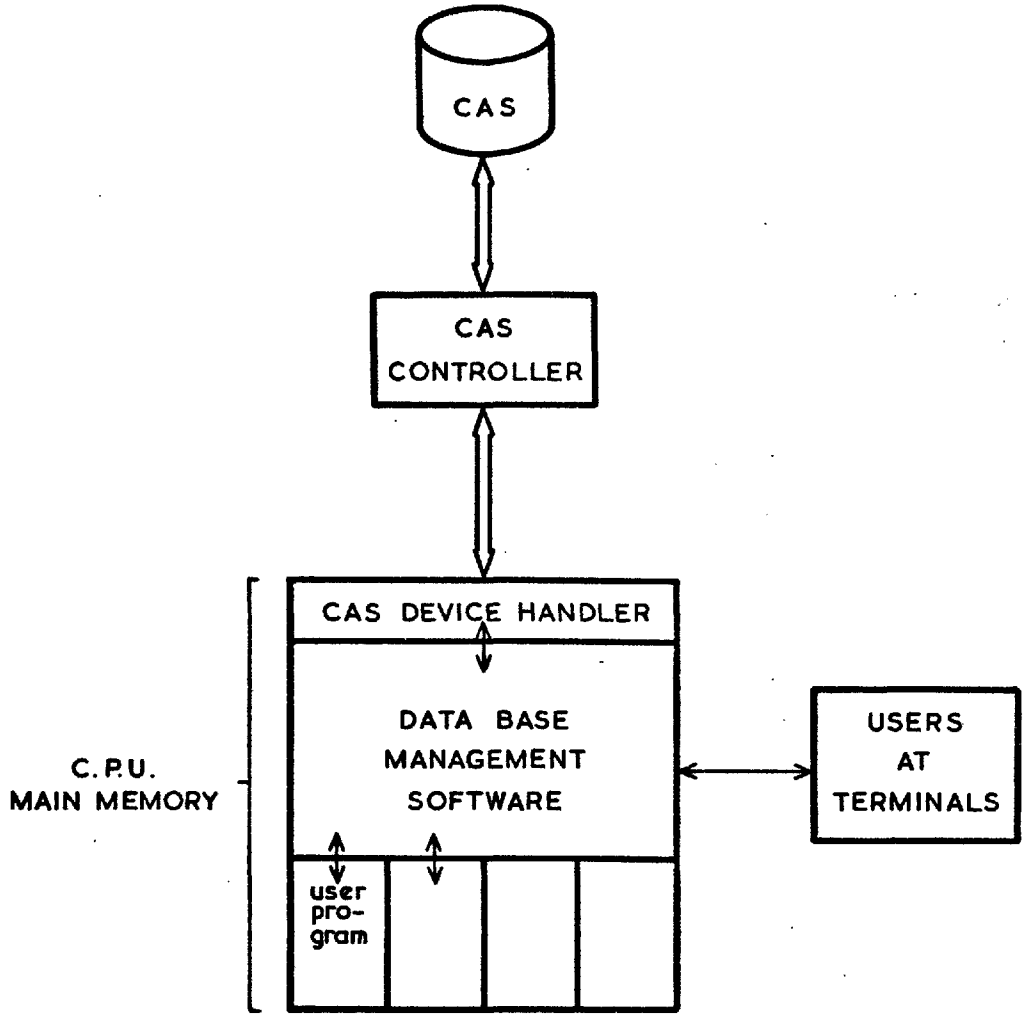


Fig. 3.5 Levels of Software

Section 3.2 describes the CAS Controller and device handler. Section 3.3 describes the interpretive language ECAL which provides an initial user language. An ECAL interpreter might replace the Data Base Management Software of Fig. 3.5 in early investigations of the CAS. An investigation of the likely performance of the CAS is provided in Section 3.4 and a discussion of hybrid systems together with some qualitative design points appears in 3.5.

Section 3.2 Design of a Content Addressable Store (CAS)

3.2.1 Basic Device Design

3.2.2 CAS Controller Features

3.2.3 CAS Data Formats

3.2.4 Functional Specifications

3.2.5 CAS Device Handling Software

3.2.1 Basic Device Design

The main performance goals derive from the survey work documented in Appendix A and listed in 2.7, these are reiterated below:-

Sizes of Data Bases:-

Useful Data Base sizes are in the range 10^8 - 10^{11} bits.

Record Sizes:-

Data base records are variable length, containing variable, repeated and optional fields. The expected record sizes are in the region 100 - 800 bytes although individual records or all records for certain applications may require much more storage.

Transaction Handling Capacity:-

Up to 400 terminals must be supported in certain cases. The highest transaction rates of which the author is aware on a single data base and using current hardware is in the region of 20/sec. (A transaction in this context meaning access to an individual Data Base record) However, rates considerably in excess of this are desirable.

Response Requirements:-

The response time of a transaction processing system is built up from different components and may include message switching and data transmission. If we define response time as the time taken to locate an individual record (or records) after being supplied with a selection criterion which uniquely identifies it (or them), then response times of about 1-2 sec. seem acceptable.

Another important requirement of any content-addressing system and one which should not be overlooked is the demand for efficient sequential access to a number of records ordered on certain of its data items. This point was considered early in the design and is discussed later, in 3.5.2.

Alternative Design Approaches

The storage capacity range suggests the use of some rotating magnetic storage medium which would have to be swept by a search mechanism and comparison logic to evaluate selection criteria. Within this context two alternative approaches present themselves.

(a) The comparison logic could be placed on each read head of the search mechanism to examine the data directly. This constitutes a parallel search on many streams of information (Multi-Stream Device).

(b) The comparison logic is held in a single controller. The information from the read heads is multiplexed and passed in a single stream through the controller (Single-Stream Device) which then performs parallel evaluation of a number of search criteria.

The physical realisation of the approach (a) above is a head per track disk with a logic chip attached to each head and a controller broadcasting a bit string for comparison to each head. Proposals, for such a device have been made by Parker[38] utilising the head-per-track Disks of the ILIAC IV machine. A brief description of the proposed device is given in Appendix A.2 its primary disadvantages are:-

(i) To evaluate a selection criterion involving multiple record items (as in the example cited in 3.1) the whole disk must be scanned many times (one for each item). During this multiple scanning intermediate comparison results are stored in bit maps within the records on the disk.

(ii) Whole record transfers take place between heads and cpu when retrieving information. Since only one head can be transmitting, multiple cycles can be required depending on the hit record distribution.

(iii) The maximum transaction processing rate is limited by the rotational speed of the disk.

(iv) Sequential processing ordered by data content is impractical using the content-addressing facilities.

The main advantage which the multi-stream device has is that the response time to retrieve a single record using a search criterion involving only a single item (as in the case for a single index) is shorter than for the single stream device.

The single-stream device was chosen by the author because it eliminates those disadvantages listed above at the expense of a longer time to search the entire store. The prime strength of the single stream device lies in its parallel task handling capability (each task involving a search criterion), impossible with the multi-stream device. This, combined with a modular hardware structure, allows the device to be tailored to user requirements trading off cost of hardware modules with task parallelism (hence transaction capacity) to achieve the desired cost/performance compromise. Further, increased performance could be attained by tactical hardware upgrading should demands on the device change as user requirements evolve.

The CAS and Supporting Software

The CAS provides content-addressing retrieval facilities for an internally stored collection of records by its "search tasks". In addition it must be able to perform conventional store-management activities for inserting new records and reorganising storage. Store management facilities are provided by two additional types of task read tasks and write tasks. For these tasks the CAS operates as a conventional direct-access store transferring a vector of information to or from a particular address on CAS storage. An initial hardware implementation of a CAS might consist of a head-per-track disk augmented with head multiplexing hardware and a controller with greatly increased logical capabilities. The original direct access capabilities would provide the direct access facilities for the write and read

tasks. The descriptions of the CAS controller appearing later will concern the hardware for content-addressing without reference to the standard direct access hardware.

The structure of the CAS controller depicted in Fig. 3.6 derives from the different steps involved in performing search tasks. Selection criteria comprise a number of "triples" (e.g. AGE < 25) connected by boolean operators "AND", "OR" and since each search task has a "selector" representing a selection criterion and a "retriever" the following operations are involved against each record in the CAS:-

(i) A set of independent comparisons of stored values with the record field(s) corresponding to a particular item.

(ii) A logical expression evaluation involving the results of the comparisons in (i) above.

(iii) The retrieval of specified record fields if the expression of (ii) above evaluates to "true".

These three activities can be associated with different units of the CAS controller indicated in Fig. 3.6. The Key Store Unit and Comparator Unit perform operation (i) above, the Search Evaluation unit performs operation (ii) and the retrieval unit performs operation (iii). During an exhaustive scan each record passes in turn through the CAS controller, the Retrieval unit copies the relevant fields into its own store and the whole record passes into the comparator unit. This then compares the record fields with values in the Key Store unit and passes the comparison results ($=, <, >, \neq$) on to the Search Evaluation Unit (S.E.U.) which is programmable. A control program in the S.E.U. performs the logical evaluation of the comparison results and if successful directs the Retrieval Unit to transmit its contents to the cpu.

A division of search tasks into <u>single-hit tasks</u> and <u>multi-hit tasks</u> is made for reasons of efficiency. It is argued that accesses to the CAS might naturally divide into two classes, either where access to a particular individual record is required (single-hit tasks)

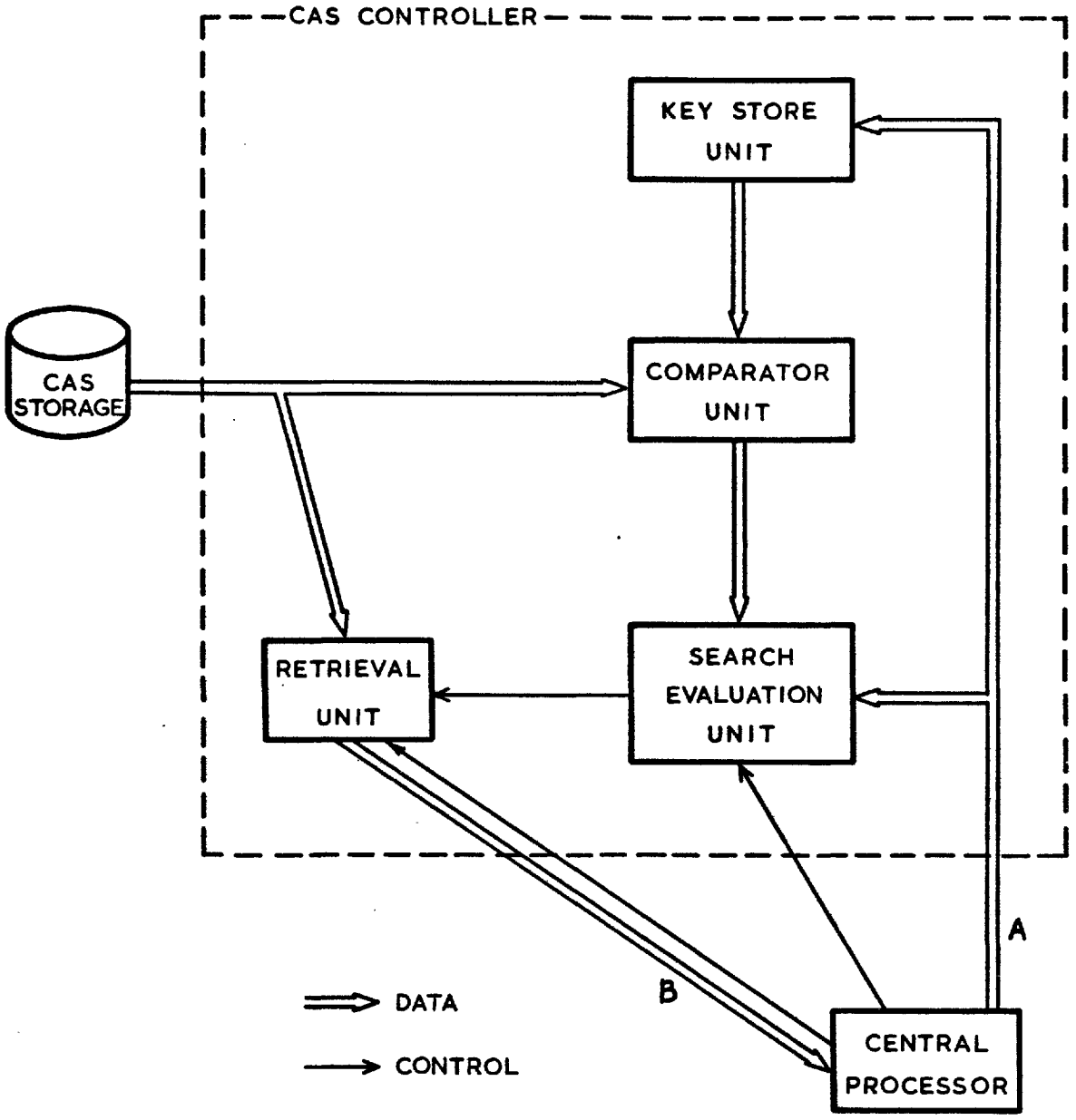


Fig. 3.6 CAS Controller Units

or where the number of records addressed is indeterminate (multi-hit tasks). The efficiency aspect concerns the time which is required to complete the task, multi-hit tasks must be operative for one entire scan of all CAS records, single hit tasks can be deleted after the first hit occurs.

The CAS supporting software comprises a software control system (Device Handler) resident in c.p.u. main memory and a suitable store management scheme for handling the CAS store allocation for its variable records. Details of the latter are not included in this thesis since they are likely to be conventional in nature, using the direct access facilities of the CAS controller.

The units of the CAS controller are described in more detail in 3.2.2. Data formats in the CAS storage are specified in 3.2.3. Specifications of the tasks which can be performed by the CAS controller appear in 3.2.4 and the CAS Device Handler (see Fig. 3.5) is described in 3.2.5.

3.2.2 CAS Controller Features

The CAS controller consists of a number of units as indicated in Fig. 3.6. Two communication channels are provided between Controller and cpu enabling the issuing of controller commands (via channel A) to proceed independently of data retrieval (via channel B). The basic parts and functioning of each unit is described with respect to the processing of a single CAS record, this is repeated for each record involved in a serial scan of the CAS.

Although the description of this sub-section relates strictly to a class of devices, "typical values" are included for certain design parameters to provide a feel for the device as initially conceived.

Key Store Unit (K.S.U.)

This unit comprises a number of key registers (32 in the current CAS proposals) which each hold an identifier, length and value corresponding to the different parts of a record field (see "Field Format" below). Also associated with each key register is a comparison result indicator containing four bits corresponding to the comparison results:- equal, greater than, less than not equal. This identifier, length and value are derived from "item name" and "value" of selection criteria. The K.S.U. contains the "values" from the selectors of all search tasks currently executing. Before a record is scanned the comparison result indicator is cleared. Each record field is then processed in sequence. The field identifier and length are compared with the identifier and length of all the key registers in the K.S.U. According to the result of these comparisons the appropriate bits are set in the corresponding comparison result indicators. After completing the processing of all record fields the contents of all the comparison result indicators are transferred to the S.E.U. for use during the execution of its stored control programs. (It should be noted that more than one bit may be set in a comparison result if repeated fields with the same identifier occur within the record).

Key registers are loaded via the LDK or LSK functions (see 3.2.4.3). A key register may be involved in more than one control program in the S.E.U. if desired - this allows the scheduler (see 3.2.5.1) to optimise the use of key registers as part of scheduling policy if required.

Substitution Key Registers

A small proportion of the key registers in The Key Store Unit are augmented to compare two fields within the same record. This has been done to accommodate "cross-coupled relations" (see 3.1 page 47). These registers have two identifiers l_1 and l_2. Initially no length or value exists in the register but l_1 and l_2 are given. When a field identified by l_1 is detected its length and value are loaded into the register. Thereafter, the register behaves as a normal key register using l_2 with the length and value recently loaded. The use of a substitution key register implies a knowledge of the sequencing of fields l_1 and l_2 in the records.

Comparator Unit

This unit contains a number of general purpose comparators which are used to perform the comparison operations between the incoming record fields and the key registers in the key store. For key registers and fields with matching identifier and length the comparator sets a single bit in the "comparison result indicator" according to the following:-

Equal	-	bit for bit equality
Greater than	-	signed integer
Less than	-	signed integer
		Not Equal - any inequality

Retrieval Unit

The retrieval unit consists of a buffer specifying the identifiers of those fields required for retrieval by all search tasks and a marker to indicate whether the record CAS address (see later) is to be retrieved. Appropriate storage is provided for retrieved material. As the incoming record passes through the controller

the retrieval unit copies the identified fields into its own storage maintaining their ordering as in the record. Later, the S.E.U. may direct the retrieval unit to transmit its contents to the cpu, appended with "task hit map" (see S.E.U.) and if required, the CAS address. This constitutes a "result" (see 3.2.3). It should be noted that a result contains the sum of the fields required for retrieval by all the active tasks. The processing of results is described in 3.2.5.3.

S.E.U. (Search Evaluation Unit)

At the end of a record scan, during which time comparison operations with key registers have been performed and retrieved material has been placed in the retrieval unit, the Search Evaluation Unit must now process the "comparison result indicators" to decide whether to transmit the retrieval material to the cpu. To enable the evaluation to overlap the processing of the next record, comparison results are transferred from the K.S.U. into the S.E.U. The S.E.U. is programmable with a program store and a pushdown stack as its main components, additionally there is a 32 bit Task Hit Map register and an addressable store for holding comparison results from the K.S.U. Each search task has a separate control program and is associated with a particular bit in the task hit map which it is able to set. The bit number (0, 1...31) identifies the task and is called the task number. There are two classes of program instruction, a test and boolean instruction described in 3.2.4.4. The former tests comparison results for a particular condition and pushes the value "true" or "false" in the top of the stack. The boolean instruction performs a boolean operation (AND, OR) between the top two entries in the stack replacing them with the resulting true or false value. The control program is a representation of a selection criterion in reverse polish notation involving boolean and test instructions, the latter addressing these comparison results corresponding to the "values" of the selection criterion. An example of a control program is provided in 3.2.4.5.

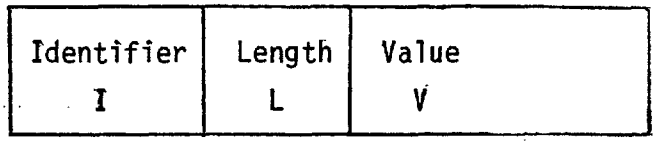
After the comparison results are read into the SEU the task hit map is cleared, then each program segment is executed in turn yielding a value "true" or "false" which is stored in the corresponding bit in the task hit map. Having executed all program segments the task hit map indicates whether the record was a hit for any task (or tasks). If so the task hit map is appended to the contents of retrieval unit and the whole is transmitted to the cpu, this constitutes a result (see 3.2.3).

3.2.3 CAS Data Formats

Information in the CAS exists as an unordered collection of formatted variable length records recognised by the controller hardware when performing search tasks. The physical organisation of these records within the store does not affect the search tasks - this is a valuable asset since different strategies for storage allocation and garbage collection may be implemented independently of the content-addressing capabilities. Physical organisation is discussed after the following description of record format.

Field Format

The smallest unit of data recognised on the CAS is the field. Records (described below) are built up from fields. Each field is a variable length byte vector consisting of three parts as follows:-



Identifier: This identifies the field type for comparison purposes. (typically taking values 0 to 255)

Length: This determines the length in bytes of the following "value" vector.

Value: This is the data component of the field and is a variable length byte vector (typically of up to 30 bytes)

Record Format

There are three classes of storage area in the CAS described later, a record constitutes one class. Records are storage vectors

consisting of a variable number of fields. Since fields are self identifying, their ordering within the record is not significant to the CAS Controller. Further, there is no hardware constraint on the relative values of the field identifiers - implying that the system software can arrange optional occurrences of fields, or repeated occurrences of fields with the same identifier. The only exception to this rule is the leading field in the record, which has special significance to the hardware as indicated below:-

T = 1	L = 2	V = record length

T: This value distinguishes record areas from other areas.

L: This gives the length of the V vector.

V: This gives the length of the whole record in bytes (typically up to 4096).

The freedom in assigning identifiers to record fields can be used in many different ways, one such way is implied by the language ECAL described in 3.3.2. If the CAS is used in conjunction with a DAS as described in Section 3.5, then each record will contain a"DAS address" for the corresponding record in the DAS. This address will occupy a normal field (or fields) in the CAS record. The contents of a DAS address is determined by the system software and may vary for different DAS hardware configurations.

CAS Address

The CAS is byte addressable. A register (typically 32 bits) in the controller gives the CAS address and this is incremented continuously as the scan proceeds. An auxilliary register is used to hold the value of the CAS address register at the beginning of each record for later transmission to the cpu if required.

Result

A result is the unit of transfer between CAS Controller and cpu for retrieval purposes, it contains information from one "hit record". Generally there are three parts to a result:-

(i) The Task Hit Map (see 3.2.2 "SEU")

(ii) The CAS Address and leading field of an area (length information) (see "Physical Organisation" below)

(iii) A list of record fields.

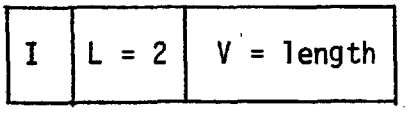
The task hit map is always present, it indicates to which of the currently active search tasks this result is relevant. Either or both parts (ii) and (iii) may be present. Part (ii) will be present if a marker has been set in the retrieval unit to retrieve CAS address and record length. Part (iii) consists of all the fields requested by all active tasks irrespective of which tasks scored the hit (indicated by the task hit map). The processing of results is described in 3.2.5.3.

Physical Organisation

The CAS is divided into areas of which there are three classes:

Free Area
Record Area
Faulty Area

Each area is variable length (in bytes) and subdivided into fields. The first field distinguishes the different classes of area and gives the area size as follows:-



Free Area: I = 0 (typically V 2^{16} - 1 bytes)

Record Area: I = 1 (typically V 4096 bytes)

Faulty Area: I = 2 (typically V 2^{16} - 1 bytes)

In the case of Free or Faulty areas no further field structure is recognised within the area. For Record Areas the field structure is recognised and takes the form previously described.

The organisation of the CAS into areas permits a variety of store management techniques to be implemented.

The various alternatives will not be discussed here since they do not influence the basic CAS design, indeed different techniques might be used in different CAS applications. The features of the CAS likely to aid store management are:-

(a) Hardware recognition of variable records.

(b) Easy record deletion simply by changing the value of 1 in the leading field.

(c) The content addressing capabilities can be used in searching for a free area of a particular length.

(d) The existence of an intelligent CAS controller permits hardware - assisted garbage collection. Halatsis [17] outlines a possible scheme.

3.2.4 Functional Specifications

The CAS controller is capable of carrying out certain tasks simultaneously. This section describes the way in which tasks are specified without regard for their parallel execution which is discussed in 3.2.5. There are three basic classes of task search tasks, read and write tasks. Only in the case of search tasks are the full hardware capabilities of the CAS controller used (i.e. key store, S.E.U. etc). When a read or write task is issued to the CAS controller it is executed immediately and exclusively, no other tasks being executed concurrently. When a Search task is issued to the CAS controller it is loaded immediately but is executed in parallel with other search tasks. The hardware mechanisms for starting and stopping search tasks are heavily implementation dependent and do not form part of the current Task specifications.

Task specifications occur at three levels in the system as indicated in Fig. 3.7. The software user-interface is the ECAL language described in Section 3.3, which primarily provides facilities for specifying the two types of search task at a symbolic level using the "SEARCH" Command. The search task specifications at the two other interfaces are substantially the same comprising a sequence of CAS Controller Commands which are used to form the selector and retriever. The differences between the specifications are explained later. The control program (see 3.2.2,S.E.U.) of the search task is passed as one parameter of a CAS Controller Command in a task specification. Single and multi-hit tasks differ only in the final instruction in their control programs,their structure is equivalent in terms of the CAS Controller Commands.

Task specifications at the software-hardware and software software interfaces appear in 3.2.4.1 and 3.2.4.2, the CAS Controller commands in 3.2.4.3 and the control program instructions appear in 3.2.4.4. An example of task specifications at the ECAL and software-hardware interfaces is provided in 3.2.4.5.

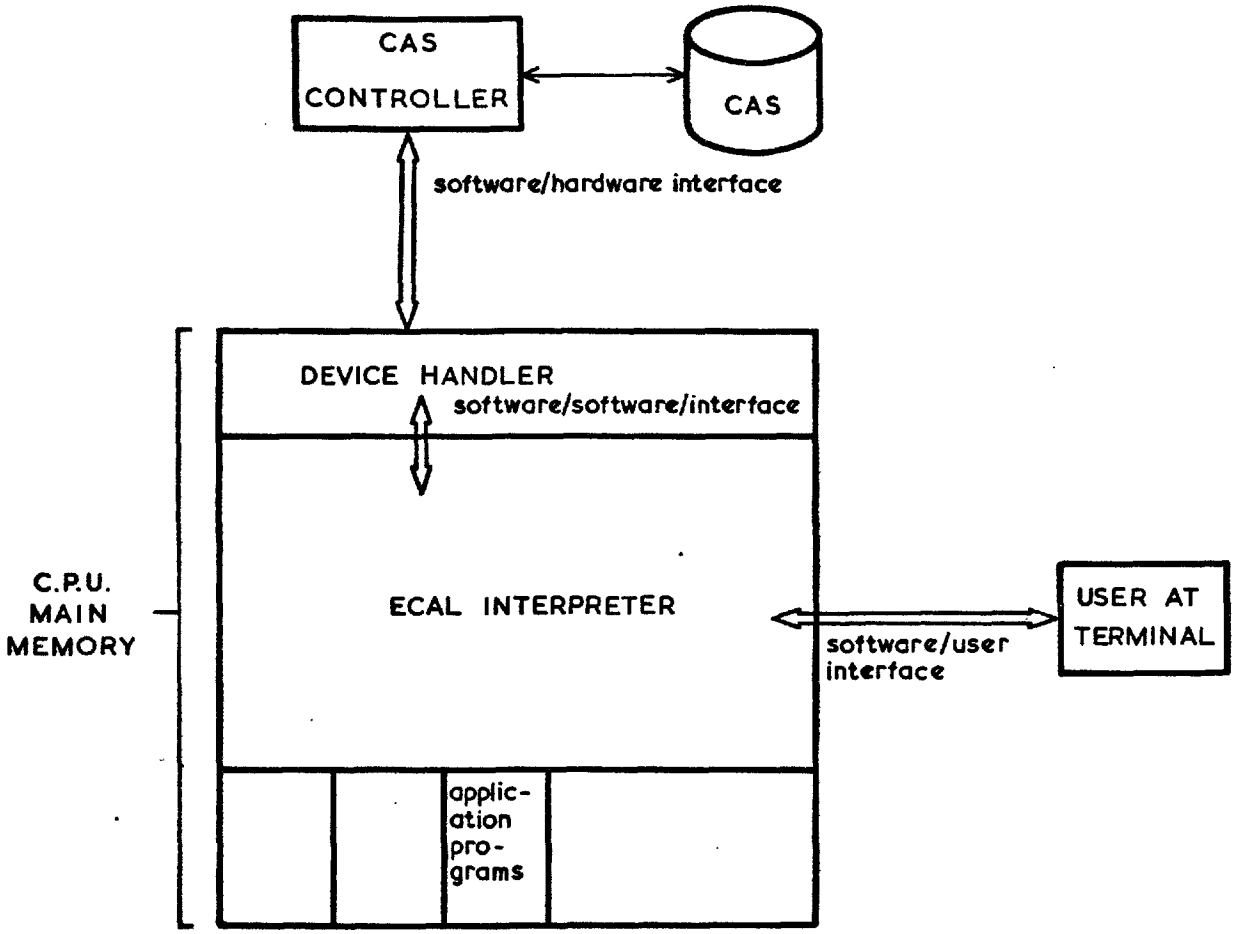


Fig. 3.7 Interfaces

3.2.4.1 Task Specifications (Software-hardware interface)

The structure of the task specifications is described in terms of the Controller Commands using a simple BNF - type notation.

Controller Commands (see 3.2.4.3)

LDK: = load key register

LSK: = load substitution key register

LRI: = load retrieval unit with I

LCA: = load CAS address marker

LCP: = load control program

WDA: = write direct access

RDA: = read direct access.

Task Specifications

<component>:= <LDK>|<LSK>

<key set>:= <component>|<key set><component>

selector := <key set><LCP>

retriever := <LRI>|<LCA> | <retriever><LRI>

read task specification := RDA

write task specification := WDA

search-task specification := <selector><retriever>

3.2.4.2 Task Specifications (Software-software interface)

The task specification issued to the task scheduler is similar to that required for the software-hardware interface except that scheduling information precedes the Controller Commands and certain parameters in the latter have not been allocated (These parameters are identified explicitly in the descriptions of 3.2.4.3).

Scheduling Information

WRT: = write task scheduling information

RDT: = read task scheduling information

SHT: = single-hit task scheduling information

MHT: = multi-hit task scheduling information

The scheduling information identifies the type of task explicitly to the scheduler and draws the distinction between single and multi-hit tasks which have the same structural appearance at the software-hardware interface. The scheduling information also identifies the local buffer (see 3.2.5) of the task and contains information enabling the unassigned parameters in the controller commands to be identified, these parameters are allocated by the scheduler e.g. key register allocations. Additionally, there may be other scheduling information like total key register requirements and control program size or priority levels etc. depending on the scheduling algorithm.

3.2.4.3 Controller Commands

A functional notation is used to define these commands where

$$F\ (N_1,\ N_2 \ldots)$$

is a function F with arguments N_1, N_2 etc. The binary representation is not defined as this will be determined by hardware requirements. The arguments will be defined as they occur.

The functions are of two classes load and execute indicated by the first letter of accronyms ("L" or "E"). Load instructions cause hardware resources to be allocated for later use by a search task execute instructions allocate hardware resources and perform the relevent task i.e. direct-access read or write operations.

Certain parameters of the load functions are inserted by the scheduler (i.e. they are unallocated at the software-software interface) . these will be highlighted in the descriptions which follow.

EWR (Execute Write)

EWR (A, E, B)

A : CAS address

E : integer indication the length of a byte vector

B : address of local buffer (see 3.3.3, SEARCH Command)

Transfers E bytes from the local buffer B to the CAS starting at CAS address A.

ERD (Execute Read)

ERD (A, E, B)

A :
E : } as for "EWR"
B :

Transfers E bytes into the local buffer B from the CAS starting at CAS address A.

LDK (Load Key Register)

LDK (K, I, L, V)

I : field identifier
L : field length } (see 3.2.3, Field Format)
V : field value

K : key register number, allocated by scheduler.

The field identifier, length and value are loaded into key register K.

LSK (Load Substitution Key)

LSK (K, I_1, I_2)

I_1 = field identifier 1

I_2 = field identifier 2

K = key register number allocated by scheduler

The substitution key register K is loaded with the identifier values I_1 and I_2 of the two fields for comparison within the same record.

LRI (Load Retrieval Unit with I)

LRI (I)

I = field identifier

The retrieval unit is loaded with field identifier I.

LCA (Load CAS Address Marker)

LCA

The marker is set in the retrieval unit indicating that the "CAS address and length" are to be retrieved.

LCP (Load Control Program)

LCP (P_i, P)

P_i = Control program (see 3.2.2, S.E.U. and 3.2.4.5)

P : Address in program store of S.E.U., this is allocated by the scheduler.

The control program P_i is loaded into the S.E.U. starting at location P.

3.2.4.4 Program Instructions

There are two main types of program instruction, a test instruction and a boolean instruction. The test instruction tests the

"comparison result indicator" of a key register and pushes a value "true" or "false" in the top of the stack. The boolean instruction performs a boolean comparison between the top two values in the stack replacing them with the resulting "true" or "false" value.

e.g. =/10: is a test instruction which pushes "true" in the stack if the equality bit is set in the comparison result indicator of key register 10. A variety of tests ($\neq$,> ,< ,$\geq$,$\leq$) can be performed in addition to the test for equality.

e.g. OR: boolean instruction performing an inclusive or between the two top values in the stack.

There are additionally two other instructions which are used to set the appropriate bit of the task hit map if the top value in the stack is "true".

SHT/T Set bit "T" for single hit task

MHT/T Set bit "T" for multi hit task

The SHT instruction additionally allows for the termination of the single hit task, informs the scheduler, the MHT instruction simply sets the required bit.

3.2.4.5 Example of Task Specification Transformation

Fig. 3.8 shows the transformation of an ECAL SEARCH command into the corresponding task specification issued to the CAS Controller.

The ECAL SEARCH Command (see 3.3.3) together with the record schema delcaration (see 3.3.2) results in the allocation of field identifiers lengths and values as indicated. The items marked with an * are additional to the user-defined selection criterion, the reason for this is explained in 3.3.3, SEARCH Command. The ECAL selector and retriever are transformed into the corresponding parts of the task specification and the items marked * are added by the ECAL interpreter. Although the task specification is substantially as produced by the ECAL interpreter certain parameter values are not allocated by the latter, they are inserted by the scheduler as

indicated. The control program is parameter P of the LCP command and is in reverse-polish notation.

ECAL SPECIFICATIONS

```
                    *                          ECAL Retriever
SEARCH   T1  FOR 'PERSONNEL' PUT ALL ' SURNAME, FORENAME, MANNO '
INTO B2 IF ,AGE LT 25 AND (SALARY GT 3000 OR POSITION EQ MANAGER),
                                               ECAL Selector
COUNT TOTAL;
```

Item	Field Identifier	Length	Value
AGE	1	4	25
POSITION	2	7	MANAGER
SALARY	3	4	3000
FORENAME	4		
SURNAME	5		
MANNO	6		
RECORD *	255	2	30

TASK SPECIFICATION (software-hardware interface)

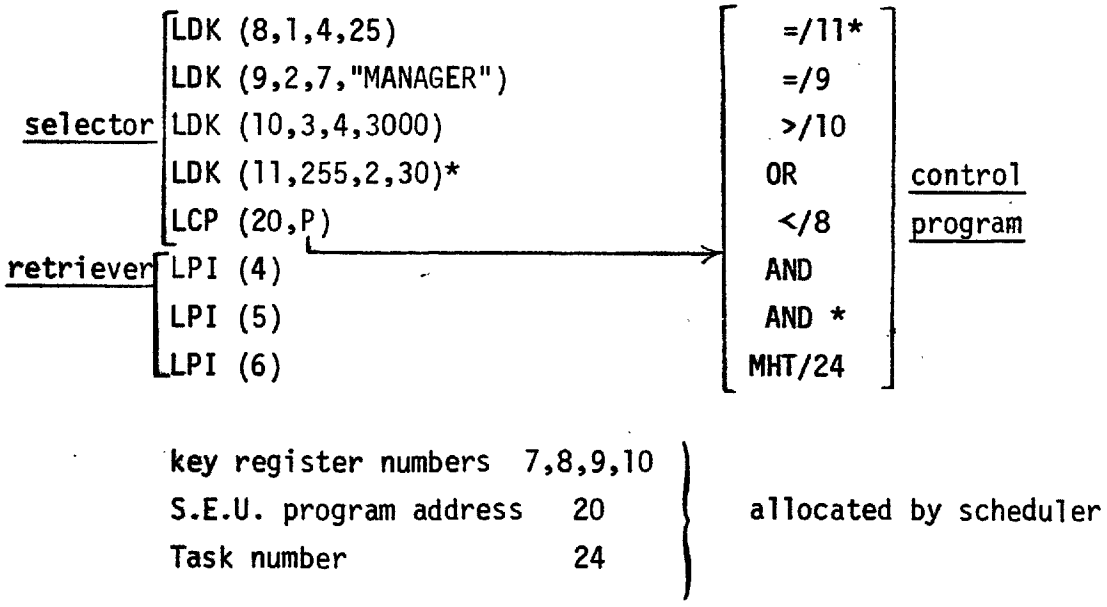


Figure 3.8 Search Task Transformation

3.2.5 CAS Device Handling Software

A valuable asset of the CAS Controller is its multi-tasking ability for content-addressing search tasks. The CAS Device Handler accepts task specifications, allocates CAS resources, controls parallel task execution and returns retrieved material to the processes originating the tasks. The relation of the Device Handler to other system components is depicted in Fig. 3.9 . There are three basic software elements in the Device Handler, the scheduler, buffer control software and the results processor. Fig. 3.9 indicates the relationship between these software elements and the data which they use. A task specification of the form indicated in 3.2.4.2 issued to the Device Handler goes to the scheduler where it is inserted into the appropriate queue. The "scheduling information" prefix identifies the task type and its local buffer (see 3.3.2, SEARCH Command) for storing the results. When resources become available in the CAS Controller the task specification is modified to the form indicated in 3.2.4.1 issued to the controller and also placed in the active task list. Data from the CAS is placed in the results buffer chain. When a results buffer is full the "Buffer Control" software passes it to the "Results Processor" which distributes the results to the relevant local buffers of the active tasks. The three software elements are described in more detail below, however it should be appreciated that these descriptions are tentative since there are many alternative strategies particularly in the case of the Task Scheduler.

3.2.5.1 Task Scheduler

The task scheduler accepts task specifications of the form indicated in and may either initiate a task immediately or place it in a queue. The scheduler uses the scheduling information in the task specification to identify the type of task. Four queues are maintained, one for each task type, also an active task list contains the task specifications of those tasks

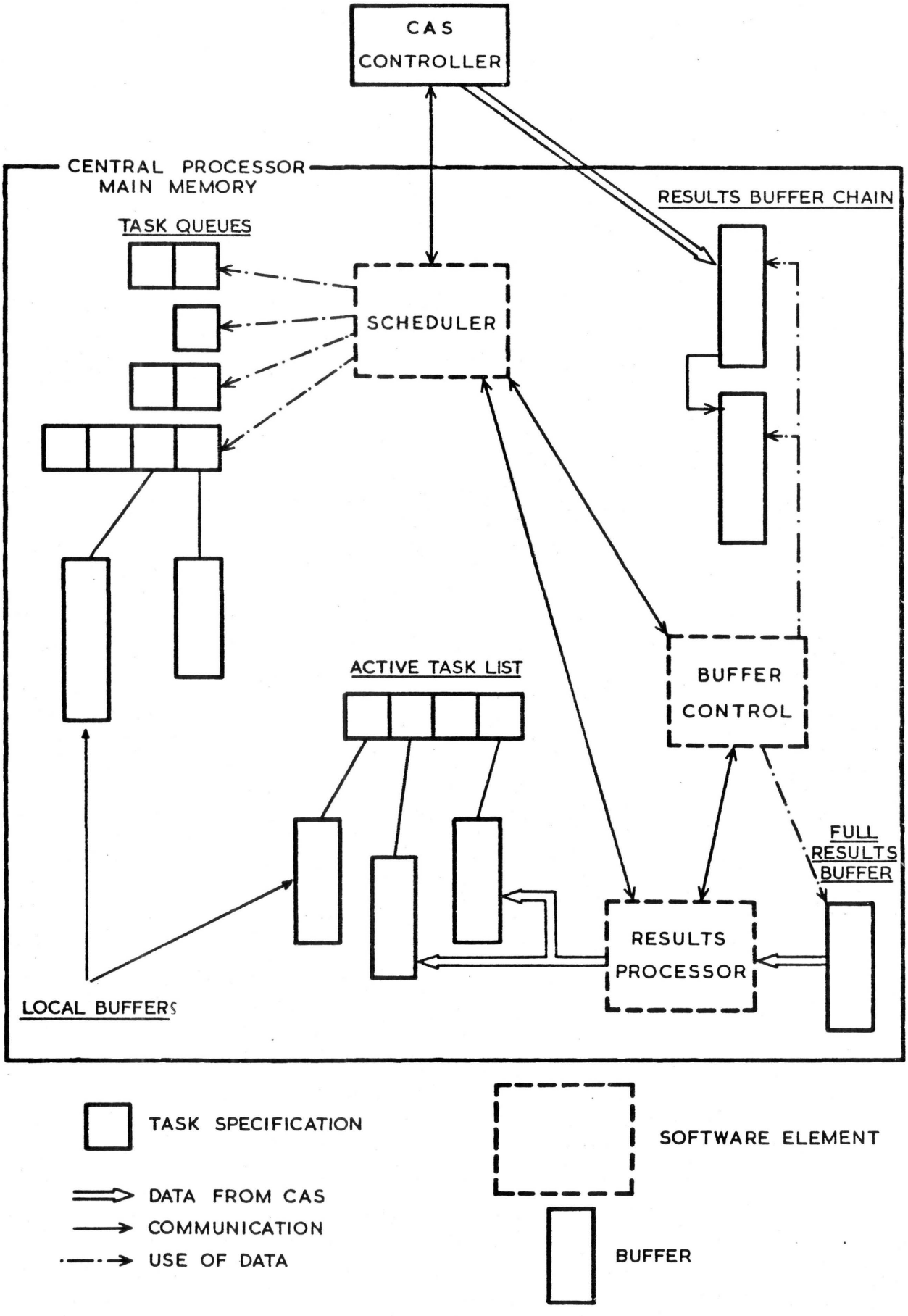


Fig. 3.9 Control Software and Data

currently active in the CAS controller. Each active task has a "task number" (see 3.2.2, S.E.U.) which identifies the corresponding bit in the task hit map of the S.E.U. The Results Processor makes use of the active task list. Within the basic framework provided by the above description the scheduler may implement a variety of scheduling policies. It is expected that scheduling policy will vary according to application requirements. One possible scheduling policy is outlined below. The aim of this policy is to provide different response times for the different classes of task, more particularly to provide rapid service for the single hit tasks.

The CAS controller scans the CAS in cycles (typically a cycle may take 1 second) where all the records in the CAS are scanned serially once only in a cycle. Queues are served on a first in - first out basis. Multi-hit tasks are loaded into the CAS Controller at the beginning of a cycle and are active until the end of the cycle then they are removed. Single-hit tasks may be loaded at any time during a cycle and terminate after the first hit, relinquishing resources which are available for re-use immediately. At the end of the scan some single-hit tasks may still be present, there are two alternative situations. Single-hit tasks which were loaded sometime during the last scan have not had a hit and have not completed a search of the entire CAS - these must continue into the next scan. Single hit tasks loaded prior to the last scan are deleted since they have been present for at least a complete scan. The hardware resources (key registers, program store) are partitioned between single and multi-hit tasks at the beginning of a cycle, the partitioning might be altered from cycle to cycle depending on the lengths of the search task queues. Read and write tasks are performed between cycles using the CAS as a direct access store. The execution of read or write tasks will not affect any single-hit tasks remaining in the Controller. From a performance viewpoint priority has obviously been given to the single-hit tasks since these are subject to the minimum delay. If the queues are

empty a multi-hit task or read or write task will have to wait on average one half cycle before being loaded. Multi-hit tasks then take a further cycle to be completed whereas the read or write tasks use the CAS as a direct access store (this will be an order of magnitude faster than the cycle time). If no read or write tasks are performed the time between scans is negligible, service is then provided to the single hit tasks on a more-or-less continuous basis restricted only by hardware resources. A performance simulation of the queueing behaviour for single hit tasks within the above scheduling policy appears in 3.4.3. After loading, the time taken to complete a single hit task is uniformly distributed in the range 0 - 1 second (with mean $\frac{1}{2}$) since the single hit record may be anywhere within the CAS with equal probability. The execution of read and write tasks tends to degrade service of single and multi hit tasks. However, it is not expected that the frequency of these tasks will materially affect performance. Bell Telephones (see A.6) have a requirement for a peak rate of 0.5 updates/second (implying write tasks) over one half-hour period. This would appear to have negligible affect on CAS performance in terms of servicing single and multi-hit tasks. It is interesting to note that the TDMS system (which provides content-addressing using conventional hardware) was rejected by Bell because it was unable to cope with such update rates.

Scheduling policy may also be required to optimise the use of hardware resources, for example by arranging maximum sharing of key registers between tasks. This might be implemented by examining the search task queues for tasks with common "item name" and "value" pairs, these tasks would then be loaded together and key registers shared. The queue service would not then be first-in first-out.

A further requirement which could be imposed on the scheduler is that specified tasks may have to be carried out in a particular sequence, for example that a particular write task (updating) the

CAS may have to be performed before certain search tasks. Facilities of this nature could be added to the Scheduler which would then hold up execution of certain queues until the required sequencing had been performed.

3.2.5.2 Buffer Control

There are two levels of buffering within the cpu. The result buffer chain and the task local buffers. The former is a chain of cpu buffers which receives the "results" directly from the CAS controller. There is only one chain of result buffers within the cpu, this chain is addressed by the CAS controller. There are a number of local buffers one for each task known to the scheduler (queued or active). There are two alternative methods of providing local buffers:-

(i) the process creating the task specification allocates the buffer

(ii) local buffers are allocated by the buffer control software from a Central pool of such buffers.

The method described in this thesis is (i) above, chosen arbitrarily. Method (ii) above appears equally suitable - in this case the local buffer would not be introduced to the scheduler, instead the buffer would be allocated at a later stage by the Buffer Control Software.

The result buffers are filled by the CAS controller. The buffer control software is notified (by interrupt) whenever a buffer is filled, the buffer is removed from the chain and replaced. The full buffer is passed to the Result Processor which transfers the buffer information into the appropriate local buffer(s).

3.2.5.3 Results Processor

The function of the result processor is to distribute the fields from "results" into the appropriate local buffers of active tasks. The Buffer Control Software presents Result Processor with a buffer containing one or more results, the active task list

maintained by the scheduler is used in processing each result.

For each result the "Task Hit Map" indicates which of the active tasks are to receive fields. For each of these tasks the "retriever" identifies which fields are requested from the result, these are then copied into the local buffer. The local buffer only receives requested fields. The format of fields in the local buffer can be tailored for subsequent interpretation by the "process" which initiated the search task, if required.

SECTION 3.3 The ECAL Language

3.3.1 Overview

3.3.2 Data Mapping Commands

3.3.3 Task Commands

3.3.4 Examples

3.3.1 Overview

The ECAL (Extendable Content Addressing Language) provides an initial user interface to the CAS. It is a command language which may either be used at a terminal to provide a user with content addressing facilities or to execute a stored program interpretively. The Implementation of ECAL by interpreter is not described in this thesis, it forms part of a separate research activity by Coulouris et al. References will however, be made to the ECAL interpreter to show its relation to other system components.

The main aim of the ECAL language is to provide user-oriented facilities for specifying the content-addressing search tasks of the CAS. The other facilities of ECAL provide a minimal framework within which these content-addressing facilities can be used. No attempt has been made to provide comprehensive data handling facilities and apart from the content-addressing facilities, ease of implementation has been an important consideration.

The relation of the ECAL interpreter to the CAS is indicated in Fig. 3.7 page 72. It replaces the Data Base Management Software mentioned in 3.1 (Fig. 3.5) for experimental purposes although it does not offer the range of facilities or user-orientation (except for content-addressing) which characterise a DBMS. A further simplification present in ECAL is that a user may only initiate a single search task at any one time.

The Data Base provided by ECAL consists of a collection of records with no explicit inter record relations. The records are divided into disjoint classes as described later in (3.3.3) and the Data Base Description consists of a record schema describing each class. Record Schemas are created by the RECORD command. The Data Structure can conveniently be described in terms of inter and intra-record structure. The latter being extremely simple in nature - record schemas comprise an unordered collection of named variable length items, the fields corresponding to each item may occur none, one or more times in each record. The inter-record structure is associative in nature and full content-addressing facilities are provided by the ECAL SEARCH task.

Record schemas used in ECAL differ from those used in the CRH system (see 4.2.4) in two important ways. Firstly the intra-record structure is much simpler being a vector of items but the records are highly variable and certain fields may be repeated or non-existent. Secondly, there is not a notion of "current record" as in the CRH system. In ECAL there is a separation (described later) between data held in buffer storage and computational storage. Fields must be transferred explicitly from buffer into computational storage before they can be used, furthermore fields from many different records may reside in a buffer as a series of "results" (see 3.3.2) In the CRH system the input/output operations provide a single current record,the fields of which may be directly addressed, there is no user-visible separation between buffer and working store.

The language consists of a set of 'commands' which operate on 'objects' which belong to various object classes. The command may define new objects within the classes, form relationships between objects and perform various operations on these objects. ECAL may be extended with the addition of new commands by adding to the supporting software, or by storing ECAL programs which may later be called with parameters. Commands are executed in one of two modes "stream" or "program" mode. In stream mode the sequence of commands originates from some device (e.g. tape reader or terminal) and in program mode the commands come from a stored program.

ECAL has four separate categories of command which can be identified with distinct operations involved in data-processing using a content-addressable store:-

(i) Transfer of data both ways between some store area (buffer) and the CAS, this is accomplished by the Task Commands.

(ii) The form of the transferred data is different from the way data is normally stored in a c.p.u. for computation. Data Mapping Commands provide high-level facilities for describing CAS record classes (record schemas) and for transforming data between buffer storage and computational storage.

(iii) Storage Allocation and Computational Commands provide facilities for allocating computational and buffer storage and for performing computation.

(iv) The above three categories can be used either in stream or program mode. In addition there are the Stream and Program-Orientated Commands which can only be used in one or other of the modes.

The most important ECAL construct is the SEARCH command belonging to category (i) above. Other commands in this category allow records to be inserted (OUTPUT command) and deleted (DELETE Command), these operations involve some store management activities. There are two classes of Data Mapping Command, the RECORD command which defines a record schema and a group of three "transfer" commands (ADD, COPY, COUNT) concerned with transferring data between buffer store and working store. The latter three commands arise because the highly variable nature of the data returned by the CAS during a search and the fact that fields are self identifying means that the form of the data in buffer storage is unsuitable for computation. Of the two options - automatic transformation into a suitably structured working area or user-controlled selective transfer of fields between buffer and working storage - the latter was chosen for simplicity of implementation, giving rise to the three transfer commands. Categories (iii) and (iv) represent more conventional facilities for computation (including character-string processing) and programming.

A detailed description of the all commands in the above categories appears in Appendix C. The remainder of this section is devoted to a description of important facets of the Data Mapping Commands (3.3.2) and the Task Commands (3.3.3). Sub-section 3.3.4 provides a very simple hypothetical example using ECAL based on the selection-criterion cited the introduction to this chapter (see page 46). Additionally, there is an example taken from a large computer user indication the type of selection criteria currently in use (via software implementation) and illustrating the ECAL representation of these.

3.3.2 Data Mapping Commands

A user-oriented language for interacting with the CAS will have some method of allocating field identifier values for user-defined symbolic item names. The ECAL language provides this facility and in addition allows the records in the CAS to be subdivided into disjoint record classes. Although the CAS hardware regards all records as equivalent a user will generally require different record classes for storing data. A simple implementation is possible if every CAS record has a record class field with a reserved identifier, all user-defined selection criteria of the SEARCH command (see 3.2.3) are then extended by the ECAL interpreter to include an "and" operation with the appropriate record class value as indicated in 3.2.4.5. This also has a beneficial effect on the allocation of field identifiers which then become "local" to the record class. The following record schema created by the RECORD command (see Appendix C.4)

```
RECORD PERSONNEL ITEMS AGE N, POSITION A, SALARY N,
FORENAME A, SURNAME A, MANNO A;
```

defines a new class called "PERSONNEL" records (i.e. a new value, say 30 for the record class field) and for each item name the field identifiers can be allocated at will (AGE = 1, POSITION = 2 etc.) The same identifier value can correspond to different items in different records without ambiguity since the different CAS records are now partitioned because of the "and" operation with the record class field. Each item has an attribute indicating whether the corresponding field is to be regarded as a numeric or character field (a distinction not drawn by the CAS device). This distinction is drawn for computation purposes.

In 3.3.1 a distinction was made between "buffer store" and "computational store" because the data representations were different. A buffer store containing data is always associated with a record schema (this association being set up by the task commands). The record schema allows the fields in the buffer to be identified for subsequent transfer into computational store and ensures that numeric and character fields are distinguished. A buffer may contain either

one copy of a CAS record (for insertion into the CAS store via the OUTPUT command) or a series of one or more results from the execution of a SEARCH command. A result contains the subset of record fields identified by the retriever (see 3.3.3) of a SEARCH command and which derive from a single hit record. The detailed formatting of data within buffers is concealed from the user. Fields can be inserted into an OUTPUT buffer by the ADD command which deals with buffer formatting. For buffers associated with SEARCH commands COUNT and COPY commands allow fields to be counted (since repetition is possible) either within a result or across all results and individual fields copied into computational storage.

3.3.3 Task Commands

There are three types of task which can be initiated using the ECAL task commands (described more fully in C.3):-

SEARCH
OUTPUT/WRITE
DELETE

The SEARCH command is used to initiate a content-addressing retrieval operation and it corresponds to the search tasks which can be performed by the CAS controller (see 3.2.4). An OUTPUT command is used together with a WRITE command to insert a new record in the CAS, this requires a CAS Controller "Write task" (see 3.2.4) but the correspondence is not as simple as for the SEARCH command. A DELETE command is used to delete a record from the CAS, it requires two different CAS tasks as described later.

SEARCH Command

The form of the SEARCH command is:-

SEARCH <task name> FOR <record name> PUT {ALL}
<retriever> INTO <LIST <buffer name>>
IF <selector> COUNT <variable> ;

The SEARCH command provides sufficient information (together with an appropriate record-schema) to construct search task specifications

to be issued to the Device Handler and with minor modifications passed on to the CAS controller. The different parts of the search task and their interpretations are as follows:-

SEARCH < task name >	-	identifies the task
FOR < record name >	-	identifies record schema
PUT {ALL}	-	the optional ALL distinguishes multi and single-hit tasks
< retriever >	-	identifies the fields for retrieval and is transformed into the search task retriever
INTO < LIST < buffer names>>	-	defines the (local) buffers to be used for the results of the task
IF <selector>	-	provides the selection criterion and is transformed into the search task selector after one modification described below
COUNT < variable >	-	the ECAL interpreter inserts a value in the variable giving a count of the number of results in the buffer on task completion.

The task name is used in subsequent COPY operations to access fields of the task results. The record schema is used to interpret the item names which appear in the Selector and retriever and is also associated with the buffers to enable the COPY operations to access result fields corresponding to individual items. Single or multi-hit search tasks are distinguished by the absence or presence of "ALL". The buffer names identify the local buffers of the task. The relation of the local buffers to other system components is indicated in the description of the Device handler and in Fig. 3.6.

The selector and retriever of the search command are transformed into the corresponding components of the search task except for a modification arising from the partitioning of CAS records into different record classes as mentioned in 3.3.2. Each record has a "record class" field reserved by the ECAL interpreter and used to distinguish the different classes. Field identifier 255 is arbitrarily chosen and reserved for this field. The user defined selection criterion must be extended by ANDing with the appropriate value of the record class field. An example of the transformation of the ECAL search command into a search task specification is provided in 3.2.4.5, the additional record class field and attendent "AND" operation are also shown.

OUTPUT/WRITE

In order to insert a new record into the CAS two steps are required:-

(i) the record must be built up in a buffer.

(ii) the contents of the buffer must be written to a particular CAS Address.

Step (i) involves the use of Data Mapping commands after using the OUTPUT command which associates a buffer with the record schema (see 3.3.2) describing the record class to be created. Data values from computational storage can then be inserted into the buffer as record fields using the ADD Data Mapping command until step (i) is complete. The WRITE command then performs step (ii), the physical transfer of the record from buffer to the CAS. An example of steps (i) and (ii) may be found in 3.3.4, Example 1, Part 2. A store-management function is embedded in the WRITE command since it has to determine the appropriate CAS address for inserting the record before initiating a CAS controller write task.

DELETE

A record can be deleted from the CAS only after it has been identified by the user, a selection criterion performs this

identification. The command takes the form:-

DELETE <selector> FROM <record class> ;

The selector takes the same form as in the SEARCH command and consequently may identify more than one CAS record however, the DELETE function is defined to delete one instance of the records. The DELETE function also performs store management operations and is carried out as follows. A single hit task is initiated using the selector and retrieving "CAS address" and "length" (see 3.2.3, Result). This CAS address is used to initiate a write task which deletes the record simply by changing the value of I in the first field (see 3.2.3, Physical Organisation).

3.3.4 Examples

Example 1

This example is based on the PERSONNEL record class indicated in the introduction to this chapter. It consists of three parts which appear in Fig. 3.10 and Fig 3.11.

Part I

Uses the BUFFER and RECORD commands to allocate buffer storage and defines the record schema for the PAYROLL records which are processed in Part 2 and 3.

Part 2

Inserts a new record into the CAS. The OUTPUT command associates a record schema and a buffer with the task. The ADD command builds up the record and the WRITE command performs the output. The buffer is freed from the task by the CLEAR command and the task name "EX1" is then re-usable.

Part 3

The SEARCH command is used to retrieve "SURNAME", "FORENAME" and "MANNO" from every record which satisfies the selector and place them in buffer.B2. The DISPLAY command is used to print the number of results and then to print the results in tabular form headed by the item names. The first result has two occurrences of the item "FORENAME".

```
--------------------------PART (1)--------------------------

BUFFER B1 (100);

RECORD PERSONNEL ITEMS AGE N, POSITION A  SALARY N,
       FORENAME A, SURNAME A, MANNO A ;


--------------------------PART (2)--------------------------

OUTPUT EX1 FROM B1 AS PERSONNEL;

ADD 30 AS AGE, "SUPERVISOR" AS POSITION INTO EX1;

ADD 1247BM AS CODE, 2300 AS SALARY, "PETER" AS
    FORENAME, "JOHN" AS FORENAME, "SMITH" AS
    SURNAME,"12734BM"AS MANNO INTO EX1;

WRITE EX1 ;
CLEAR EX1 ;
```

FIGURE 3.10 Record Schema and Record Creation

----------PART (3)----------

```
SEARCH T1 FOR PERSONNEL PUT ALL SURNAME, FORENAME, MANNO
     INTO B2 IF AGE LT 25 AND (SALARY GT 3000 OR POSITION EQ
     "MANAGER")  COUNT TOTAL ;

DISPLAY TOTAL;
     3

DISPLAY T1 ;

     SURNAME          FORENAME          MANNO
     THORPE           FRANCIS           104BI
                      EDGAR
     JONES            MARTIN            73604AI
     GILL             KENNETH           1117BI
```

FIGURE 3.11 Content-Addressing

Example 2

This example is taken from program listings of a large computer user currently making use of the ICL "FIND" package to address files by selection criteria for batch report-generation. (the package performs an exhaustive search of the files) The example appears in Fig. 3.12 and consists of two parts.

Part 1

This is the original program listing of two typical selection criteria. In the second selection criterion the "triples" are replaced by specifying a range between (BET) two values for the item.

Part 2

This is the ECAL representation of the two selection criteria. In the second selection criterion the range is replaced by tests within the two limits connected by an AND operator.

```
----------PART 1 ----------

(MONTHAV GTP 0 AND (HSGA GTR 1499999 OR HSGA LSS 1400000 ))

?AX01
(AVPRICE RET 0000000001,0000059999 AND MONTHAV RET 0000000000,0000769231 )

----------PART 2 ----------

MONTHAV  GT  0  AND (HSGA GT 1499999 OR HSGA LT 1400000)

AVPRICE  GE  1  AND AVPRICE LE 59999 AND MONTHAV  GE  0
     AND MONTHAV  LE  769231
```

FIGURE 3.12

SECTION 3.4 Performance Evaluation

3.4.1 Overview

3.4.2 Comparison with Inverted File Performance

3.4.3 Transaction Processing Queuing Model

3.4.4 Summary

3.4.1 Overview

In this section an attempt is made to estimate the performance potential of the content addressable store (CAS). These evaluations have been kept at a high level in an attempt to reflect the basic nature of the physical situations without involving too much detail which might cloud the essential issues and could in any case alter substantially during subsequent development. It was felt that two applications of the CAS could usefully be investigated. Firstly, its performance against a typical Data Base compared with an "Inverted" storage-structure. Secondly, its performance investigated in a transaction processing environment comprising a high frequency of simple accesses to the CAS.

3.4.2 Comparison with Inverted File Performance

The performance of the CAS is compared with that of an inverted storage-structure for the same Data Base. A comparison with the multilist storage structure was not undertaken because the retrieval response is, in general, an order of magnitude slower than inverted systems. It will be recalled that the CAS could be based on a conventional head-per-track disk of about 10^8 bits capacity. For a fair comparison the Data Base is assumed to be held on this class of disk for the inverted storage-structure and on the modified version of the same disk acting as a CAS. Due to the storage overheads inherent in the inverted storage-structure it is reasonable to expect the Data Base to be about half the capacity of the device, although the CAS could use virtually the entire capacity. The choice of Data Base size is not of major importance since the performance of an inverted system falls off relatively slowly with increases in this parameter.

Four aspects of performance require investigation; response times for retrieval and inserting a new record, I/O Channel usage and cpu usage. The latter is particularly difficult to estimate quantitatively. In the case of the CAS it is relatively easy to estimate the first three aspects since they fall-out naturally

from the device description (3.2.2). Lefkovitz [27] describes methods for estimating inverted storage-structure performance and derives formulae which appear to be at the appropriate high-level for our purposes and are used in the calculations appearing below. A summarised comparison of the calculations appears in 3.4.4 together with a qualitative assessment of cpu usage.

Relevant Parameters

Direct Access Store (head per track):

Capacity $\simeq 10^8$ bits
Rotational Speed 30 millisec

Database:

20,000 records
300 bytes/record (30 fields each 10 bytes)

Selection Criterion:

5 items (triples)

Record Retrieval:

10 records

Directory Structure (Inverted file):

Three level on disk

Queuing Time:

In both CAS and Inverted systems there will be scheduling activities, for purposes of this comparison queuing time before job or task execution is ignored.

Retrieval CAS

A search task takes one complete cycle.

$\simeq 1$ sec

Update time

The CAS operates in direct access mode, it is assumed that store management maintains a core-resident table of free store with varying area size so that the single write task to a known CAS address performs the insertion. The time is composed of rotational latency plus a single track write i.e. 15 + 30 msec.

$\simeq$45 millisec

I/O Channel Usage (CAS)

Retrieval:

Transmit search task specification	1 transfer
Receive results	10 partial transfers
	11 transfers

Parameters Used in Formulae which follow

Tn :- Time to decode an n-level directory tree for an inverted file

Te :- Time to read and update an address list

R :- Disk Revolution Time

Tr :- Head positioning time (zero on a head-per-track-disk)

Nt :- Number of items in selection criterion

L :- Average address list length

A :- Number of direct access addresses per physical block on the direct access store

ph :- Number of records which satisfy the selection criterion

Nk :- Number of keys per record

The values of L and A are difficult to assess individually but they only occur as the quantity L/A in the formulae which follow We will assume the value of L/A = 1, which implies that the block size is chosen to correspond with the storage required for the average address list length.

Retrieval (Inverted)

Directory Decode:

$$Tn = Tr + (2n - 0.5)R$$

$$P = 0$$

substituting $R = 30$ ms

$$n = 3$$

$$Tn = 5.5 \times 30$$

$$Tn = 165 \text{ ms}$$

Directory decode for 5 triplets = $Tn \times 5$ = 825.ms

List Intersection:

$$\frac{L}{A} Nt\ (Tr + 1.5R)$$

substiuting $\frac{L}{A} = 1$

$Nt = 5$

$Tr = 0$

$R = 30$ ms

List intersection = $5 \times 1.5 \times 30$ = 225 ms

Record Access:

$$pL\ (Tr + 1.5R)$$

substituting $pL = 10$

$Tr = 0$

$R = 30$ ms

Record Access = $10 \times 1.5 \times 30$ = 450 ms

Total Retrieval Time = 1.5 sec

Single Record Insertion (Inverted)

Directory Decode: Nk.Tn

Substitutes $Tn = 165$ ms (previous calculations)

$Nk = 30$ (keys/record)

Directory decode = 4950 ms

Inverted list read/update: $Te = Tr + 1.5R + \frac{1}{2} \frac{L}{A} R$

Substituting Tr = 0

$\frac{L}{A} = 1$

R = 30

T1 = 60 ms

Inverted list update = Nk.T1 = 1800 ms

Record insertion: Tr + 1.5R

Substituting Tr = 0

R = 30

Record insertion = 45 ms

Total Record Insertion Time ≃7 sec

I/O Channel Usage (Inverted)

Retrieval:-	Directory Decode	15 transfers
	List Intersection	5 transfers
	Record Access	10 transfers
	Total	30 transfers
Insertion:-	Directory decode	90 transfers
	Address list read/update	60 transfers
	Record insertion	1 transfer
	Total	151 transfers

3.4.3 Transaction Processing Queuing Model

A study was made of the queuing behaviour of the CAS by constructing a simulation program to model its operation in an environment comprising a high rate of simple record accesses as might arise in transaction processing applications. This class of application appears rather different from requirements established previously

for hardware evaluation of complex selection criteria for which the CAS was designed. Nevertheless, the high scanning rate and comparison parallelism inherent in the CAS appear well suited to this application although the Search Evaluation Unit logic facilities for evaluating boolean expressions would not be used to the full. It is probable that the type of configuration suitable for transaction processing corresponds to a partially inverted system where the CAS is used to hold indexing information for a much larger Data Base on direct access storage as indicated in Fig. 3.4. The CAS being used to retrieve the direct access addresses of individual records which are then retrieved from the direct access store for processing. Assuming each transaction requires access to one Data Base record a single-hit task can be generated for each transaction. The existence of two separate storage devices increases the response time for Data Base access but does not reduce the throughput. It has been observed that existing systems have difficulty in coping with transaction rates of up to 20 per second, and are limited by I/O channel capacity and disk latency deriving largely from index processing. The theoretical capacity of the CAS appears to be 60 transactions (single hit tasks) per second and furthermore, since index processing is not required by the DAS it is likely that the existing devices could cope with single record access rates approaching this figure. The figure of 60 transactions per second is certainly the upper bound on performance but it is likely that queing effects will manifest themselves before this is reached and it is important to establish the workable limits of performance. The operation of a CAS in the transaction processing environment can be modelled as an "m-server queue with uniform service-time and poisson arrivals". It is assumed that transactions are generated by a large number of essentially independent sources (terminals, parallel multi-program execution etc) and in this case Cox and Smith [41] show that the frequency distribution of transactions is poisson. The CAS Scheduler (see 3.2.5.1) maintains one queue for "single-hit tasks" and on the assumption that each transaction gives rise to one single-hit task then queue arrivals will be poisson

distributed. As stated earlier each of these tasks only requires one CAS key register. For this transaction processing problem the scheduler can be considered only to maintain the one FIFO queue, and when a key register becomes available the first item in the queue is instantaneously removed from the queue and allocated the vacant key register. Each key register is equivalent from the point of view of servicing any task and therefore the queue can be considered as having "m" identical servers (in the case of the CAS m = 30, excluding the substitution key registers). Each single hit task will address a unique record in a random position on the CAS, consequently the service time for a task will be uniformly distributed in the range 0-1 sec (a complete scan taking 1 sec) and the mean is 0.5 sec. The model is thus considered a simple representation of the physical situation, reflecting the essential characteristics.

Advice from the Mathematics Department at Imperial College indicated that the m-server model described above does not have an analytical solution due primarily to the intractible mathematics resulting from the uniform service-time distribution. The only relevant analysis available concerns a proof that an equilibruim distribution for queue length exists up to the maximum theoretical arrival rate (i.e. in the case of the CAS 60 transactions/sec). However, this is only of academic interest since the mean queue length tends to infinity at this point. A computer-based simulation was therefore undertaken by programming the model in Fortran which was adequate for this need and, moreover, readily available.

The simulation program was parameterised as follows:-

(i) Total Time for simulation

(ii) mean service-time

(iii) mean transaction rate

(iv) number of servers

The output statistics were:-

(a) queue length distribution and computer mean

(b) response-time distribution and computed mean

(c) mean key register utilisation.

A particular CAS was specified by assigning values to the parameters (ii) and (iv). (For the CAS described in this thesis these took the values $\frac{1}{2}$ sec and 30 respectively.) Parameter (iii) gave the mean value of the transaction arrival poisson distribution. Parameter (i) gave the total simulation time. The output statistics were primarily equilibrium distributions for queue length and response-time (i.e. queue-time and service time). The mean key register utilisation was included as a check on the correctness of the model since its value should linearly increase from 0 to 100% as transaction rate goes from 0 to the maximum theoretical value. The simulation time had to be sufficiently long to allow equilibrium distributions to be attained - a situation reached when the mean queue length has reached a stationary value. The simulation time was allocated by trial and error, using values of mean queue length sampled at equal time intervals throughout the simulation to indicate whether the mean had reached a stationary value. These values were printed with the output statistics. A description and flowchart of the simulation program is provided in Appendix D.

The results of the simulations are presented graphically:-

Fig. 3.13 shows the variation of mean queue length and mean response-time with transaction rate for the CAS.

Fig. 3.14 shows how the service-time distribution for the CAS alters in form as the transaction rate increases.

The results of the simulations are summarised in 3.4.4.

3.4.4 Summary

Comparison with Inverted File Performance

Figure 3.15 summarises the quantitative results from 3.4.2 for comparison. The following qualitative discussion of cpu usage

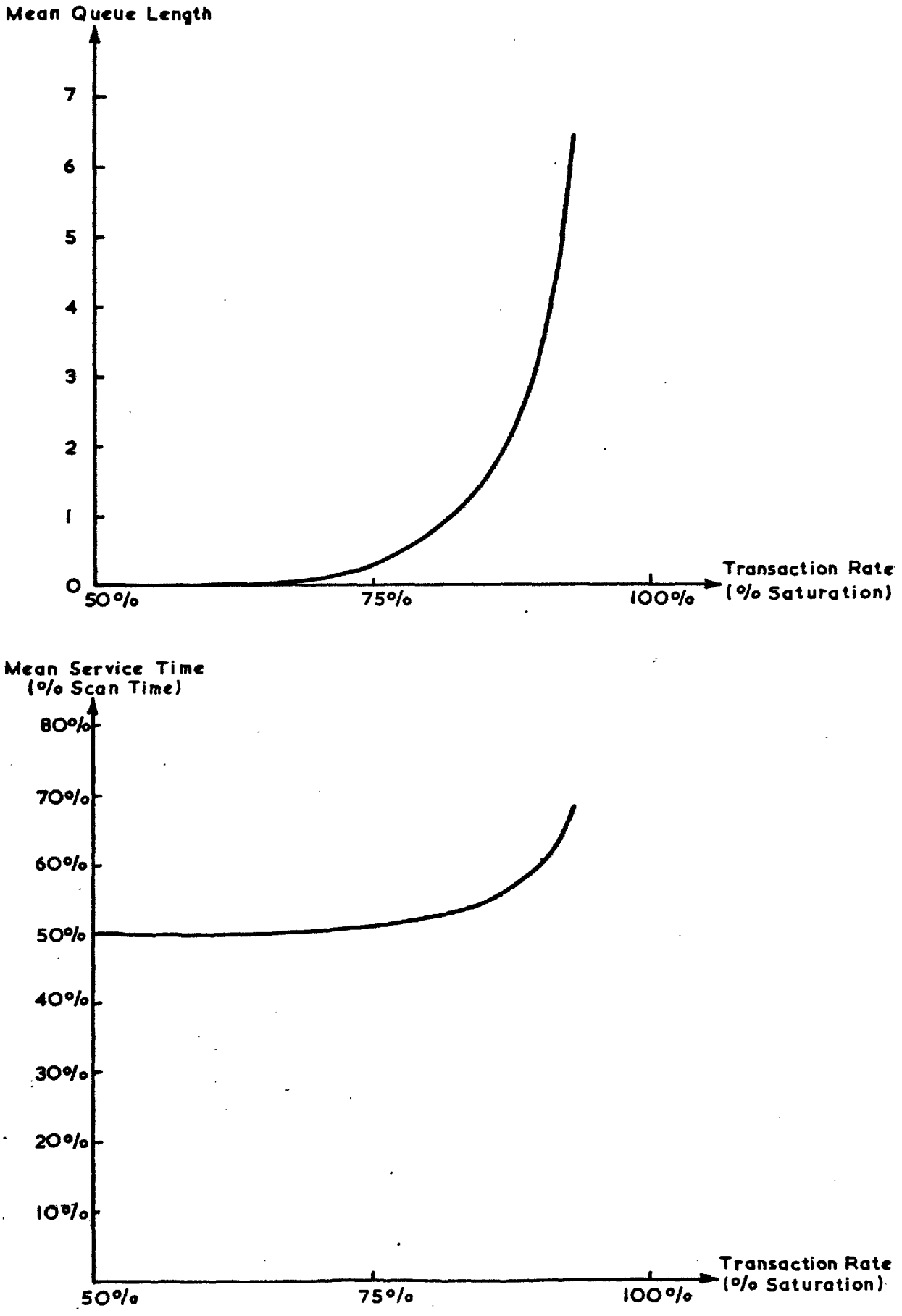


Fig. 3.13 Queue Length and Service Time Variation

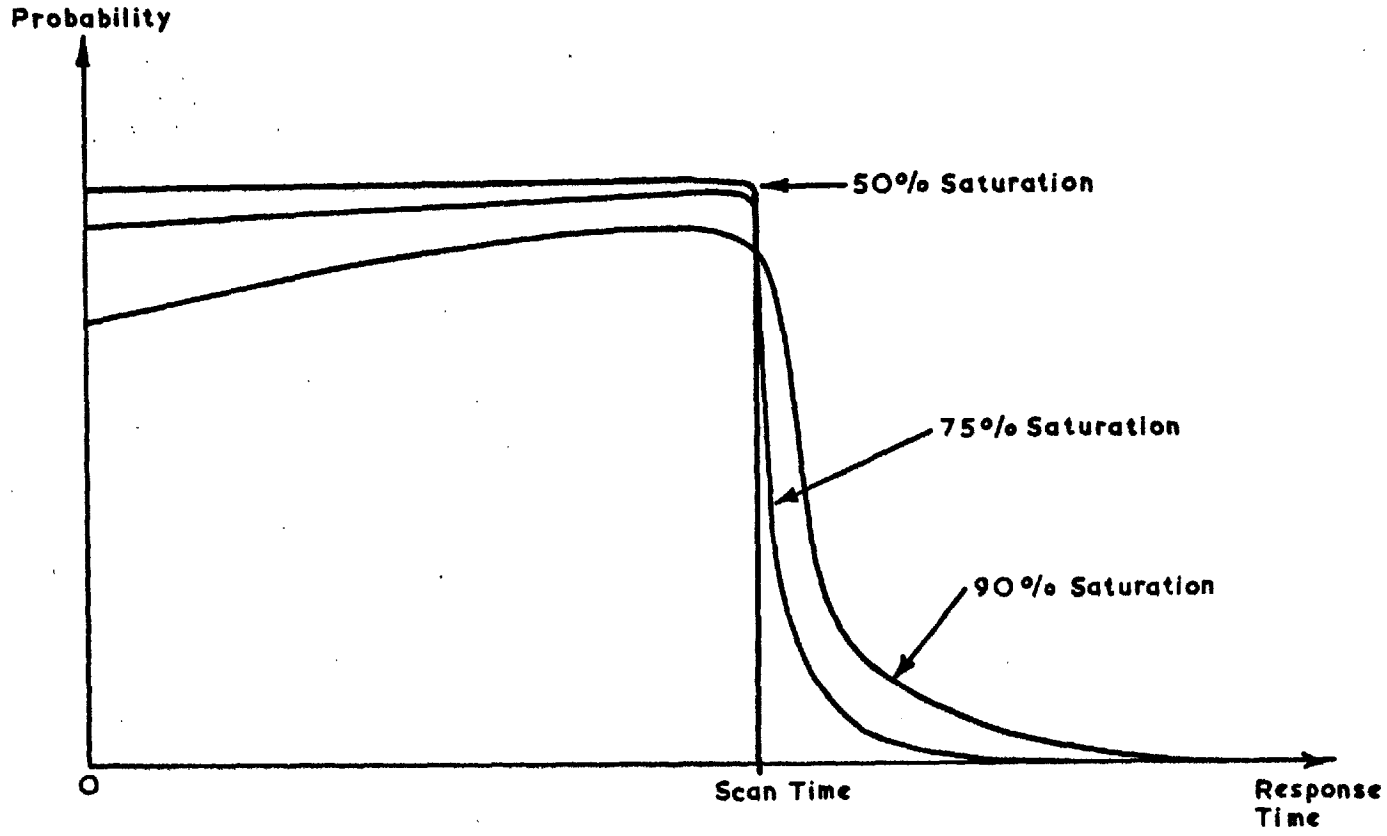


Fig. 3.14 Service Time Distribution

	Retrieval	Update	I/O Transfers	
			retrieval	Update
CAS	1 sec.	45 m.sec.	10	1
Inverted	1.5 sec.	7 sec.	30	151

Fig. 3.15 Inverted File vs CAS Performance

is put forward. In terms of main store requirement for buffering the CAS appears to have a significant advantage since buffer space is only allocated for "hit" material i.e. that which satisfies the selection criterion. Whereas the inverted system has a significant allocation of buffer storage for logical address list processing (Lefkovitz shows six buffers allocated for this purpose). In terms of processing time the CAS appears considerably better since content addressing is performed autonomously by the CAS controller but the inverted system makes heavy demands during address processing.

Accepting the initial assumptions for comparison the CAS appears to offer considerable gains over the inverted system. The assumption was that each system handles a single selection criteria is perhaps the most unrealistic since both systems have potential for parallel evaluation. Up to the maximum capacity of the CAS controller (six of the example selection criteria) there is no reduction in response-time for parallel task handling (since all are serviced in one scan). However the inverted system makes 30 I/O transfers in 1.5 sec for one selection criterion and it is likely that parallel evaluation of six such selection criteria would degrade response appreciably even allowing for some over-lapping of index requirements.

Queuing Model Results

The queuing model indicates that up to 75% loading there is no degradation of CAS performance and that up to 90% loading the response time degradation is less than 20%. From these results it appears that transaction rates in the region 40-50 per second are achievable using the CAS without causing queuing problems. (The mean queue length at 45 transaction/sec being less than 1.0.) Reliability checks on the programmed model included verifying the arrival and service-time distributions, evaluating mean server utilisation and examining the limiting case of the response time distribution - This can be seen as the rectangular distribution of Fig. 3.14 which is expected for zero queue length (serve-time = response time).

The above results should be viewed as indications of likely performance but not as hard fact since the extent of the similarity of the model to the physical situation is not established. Factors likely to detract from the similarity are:-

(i) Effects of high transaction rates introducing scheduler delays in transferring tasks from queue to servers.

(ii) Variations in mean arrival rate e.g. when a number of terminals are switched in at a particular time of day.

Section 3.5 Further Considerations

Hybrid Systems

The CAS may be used in conjunction with a variety of peripheral storage devices. An important configuration comprises a CAS together with a high-capacity Direct Access Store (DAS) as indicated in Fig. 3.16 (multiple DAS devices could be used instead of the single DAS shown). Methods of utilising both devices are discussed later. In Fig. 3.16 the CAS and DAS are said to have a loose interaction where each device has its own controller. In this case correllation between operations on the CAS and the DAS is entirely by software since each device is separately under cpu control. An alternative configuration is shown in Fig. 3.17 where the CAS and DAS both have the same controller (i.e. the CAS controller described previously with some additional features) and in this case there is a tight interaction between the two devices. The controller facilities for selective retrieval of record fields are applied to the DAS and the correllation between CAS and DAS addressing operations is recognised by the controller rather than the cpu.

The above-mentioned configurations correspond with a conventional partially inverted (see 3.1) storage structure. The DAS may contain a data base larger than the capacity of the CAS since generally only a subset of the data base record fields need be involved in content-addressing. Discussions with the users of the survey in Appendix A and the implementators of the ICL "FIND" package suggests a ratio of 1:10 for CAS to DAS capacity. The data base records are stored on the DAS and certain "key fields" are stored in the CAS. There is one summary record in the CAS for each DAS record, the summary record containing the key fields of the corresponding DAS record together with a field containing the direct access address of the latter. Selection criteria may contain only the items corresponding to the key

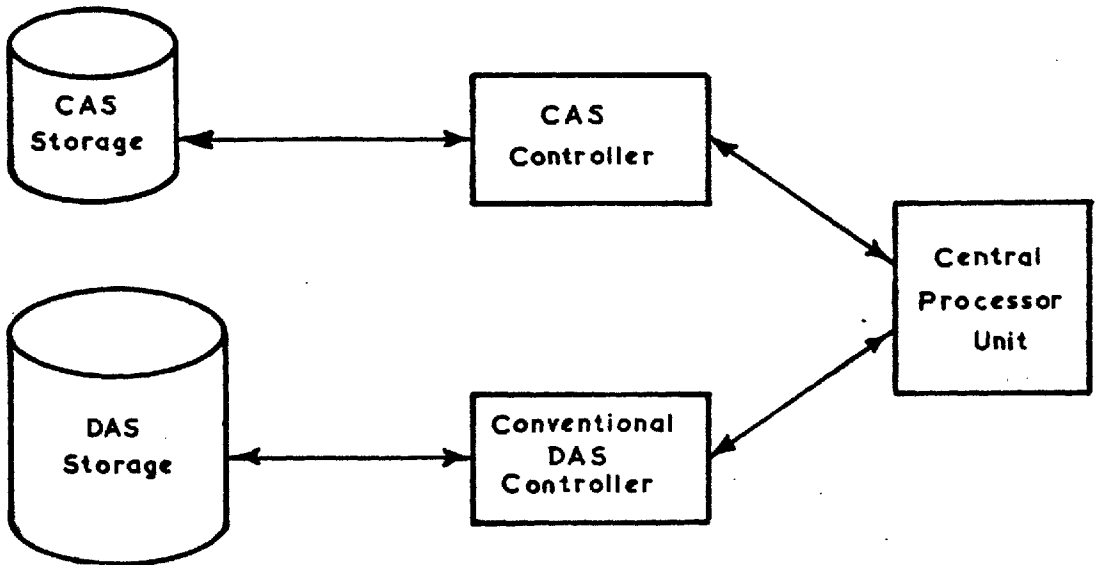


Fig. 3.16 Loose Interaction

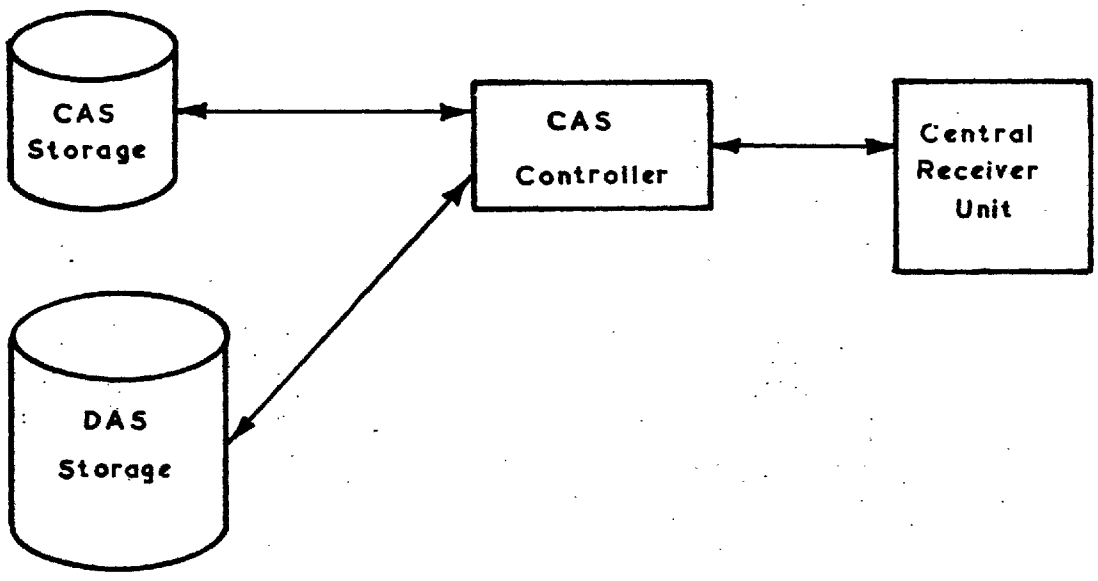


Fig. 3.17 Tight Interaction

fields. In the loose interaction configuration content-addressing operations are applied to the CAS and the direct access address fields of hit records are retrieved. The cpu then issues these direct access addresses to the DAS to retrieve the data base records required for manipulation. All user access is via the content addressing facilities of the CAS. The system software maintains the values of the DAS addresses and is free to implement any desired store management schemes. The DAS intra record structure need not correspond with that of the CAS. In the tight interaction configuration the CAS and DAS record structures are similar and the CAS controller recognises the DAS address fields of the summary records. The "selector" of a search task is applied to the CAS retrieving only the DAS address fields which are then stored in a controller buffer, each DAS address is then used together with the "retriever" of the search task to retrieve fields from the appropriate data base record.

In the early stages of development loose interaction is preferable because of the greater flexibility for experimental purposes. As an alternative to the purely content-addressing scheme outlined above both the DAS and CAS could be used for user access. Some "explicit structures" (see 2.3) could exist in the DAS, for example a scheme for network inter-record relations like DBTG COBOL or IDS might exist. The CAS could then be used to augment these network data-structures with content-addressing facilities. Under loose interaction conditions this is feasible since the CAS and controller do not make any demands or restrictions on the way data is organised in the DAS. This is particularly important since any new hardware device should be suitable for adding to existing systems without major uphevals.

Sequential Access

Efficient sequential access to data base records is extremely important. The point was made in 2.2 that sequential access facilities must be provided within the data-structure. Direct

use of content-addressing is not suitable for sequential access based on ascending or descending field values (corresponding to a particular item or items) since a selection criterion identifying a particular record required "next" for processing requires some precise information about field values which is not known during sequential processing. The method suggested for sequential processing involves the use of standard sorting techniques. If the CAS is being used alone then the fields required for sequencing together with the "CAS address" are retrieved from each record of the class to be accessed. The CAS addresses are now ordered by sorting using the corresponding field values, then the ordered list is used to sequentially access the records (using the direct-access facilities). If a hybrid system were being used the same logic would apply except that the "DAS addresses" would be retrieved, sorted then used to access the DAS sequentially. This is a practical method because the amount of material for sorting will probably be very small in relation to the total of the information in the records. If, however, this retrieved material would itself be large and unmanageable then the content-addressing facilities could be used beneficially to only retrieving certain ranges of field values. The records in the class are thus partitioned into the different ranges which are sorted independently. The above method does not require any physical sequencing of records in the CAS or DAS so that no complications are introduced into the store management of these devices. Undoubtedly there might be certain specific applications which would benefit from physical sequencing of CAS records and these might be dealt with as special cases requiring more complex store management (note: The content-addressing capabilities are entirely unaffected by such changes).

In addition to sequential access via the CAS proposed above, the DAS itself has sequential access capabilities which could also be used beneficially for batch-processing activities.

Access Synchronisation for Multi-Users

The CAS does not appear to offer special difficulties in this respect. Using the CAS as a stand-alone device synchronisation can be effected by controlling the execution of CAS tasks. The scheduling policy outlined in 3.2.5.1 provides parallel execution of search tasks but separates write tasks which are individually executed. This removes contention problems at the physical level. Minor modifications to scheduling policy could provide facilities for executing tasks in a particular sequence so that for example a search task could only be executed after a particular write task. If it were desired to allow records to be "locked" explicitly this could be accomplished by marking records appropriately. The marks could be processed either at an entirely software level by the "Results Processor" or by using the content-addressing hardware to exclude the retrievals of marked records.

In the case of a hybrid system involving a CAS and DAS the Data Base Management system software is responsible for issuing access requests to the DAS and is therefore able to implement any desired schemes.

Intra Record Structure

The CAS itself satisfies the requirements for variable and optional fields but does not have any explicit intra-record structure over and above this. It is quite probable that in a hybrid system the Data Base records on the DAS will have a different intra record structure, probably tree-like in nature (having variable and optional fields) similar to that of DBTG extended COBOL. The type of record structure in the DAS is not constrained by the CAS record structure.

Resilience

Potentially the CAS appears able to increase the "resilience" of a data base system to data corruption. In the event of data base corruption or faults developed in areas of magnetic storage the CAS appears more amenable to recovery than a conventional direct

access store where corruption to pointers can effectively "disconnect" whole chains of records resulting in repercussions far wider than the corrupted area. Not only are explicit relationships lost over a wide area but valid records may also be lost if records themselves are not readily distinguishable in physical storage. The effects of corruption in the CAS are localised, valid records are recognised by the hardware and accessible via content-addressing operations.

In a hybrid system it is likely that the duplication of data base information in the CAS "summary records" would be beneficial in the event of corruption in the DAS. The DAS addresses of data base records would still be present in the CAS, furthermore record access would still be possible via the key fields stored in the CAS.

Privacy

The content-addressing facilities can be used to provide hardware checked record level privacy locks for CAS records. Certain record fields might be used as a privacy lock. Values of the privacy lock field would be "AND"ed to all selection criteria in the same way as the record class identifier was used (see 3.2.4.5) to effect a partitioning of CAS records. The allocation of privacy key values could either be explicit or effected implicitly by the Data Base Management system.

Chapter 4 : A Commercial Record Handling (CRH) System

4.1 General

4.2 The CRH Language Facilities

4.3 Implementation of the CRH System

Section 4.1 General

This chapter describes a "host-language" Data Management system called the "Commercial Record Handling" (CRH) system which provides format-independence for compiled programs as its main feature. As defined in 2.1 format-independence is the provision of data independence only for intra-record structure. The CRH system extends the procedural host language (in this case Basic Language) with data manipulation and data description facilities which provide primarily high level descriptions (record schemas) for intra-record structure. A program compiled to operate on records with a particular structure is able, without recompilation, to successfully process records with a different structure provided certain conditions are met i.e. the program is retrieving information from existing records and the program declared record schemas are valid sub-descriptions (or sub-schemas , see later) of the actual record structures. The binding of program and data is a run-time activity performed once only for each record class, it is not performed dynamically for each instance of the records in a class. This is possible because record structures are largely fixed within a class. Data processed by the CRH system exists in separate files each containing a number of distinct records accessed sequentially. Files may contain a number of different record classes, the members of which may be freely interspersed in the sequential storage medium. Each file has a "header" which contains a coded description of those record classes on the file. Symbolic programs include record schema descriptions which are stored in a compact coded form in the object program after compilation. When a file is opened the data is bound to the program using the program record schemas in conjunction with the file header.

The CRH system exhibits an important characteristic of Data Base Management systems in that the data itself has a separate description which exists independently of processing programs. However, the

simplicity of the sequential record access facilities (or inter-record structure) preclude its classification as a full Data Base Management System. A preferred viewpoint is that the CRH facilities are an enhancement of a simple File System to provide format-independence for record processing and high-level record class descriptions (record schemas). As such it does not easily fall into any particular category from the classification of 2.2.1. However, this classification arises from the observed features of existing systems which have evolved over a period of time - it does not necessarily imply that user requirements can be neatly compartmentised in an exactly similar fashion. In particular, smaller batch-processing users with relatively simple data-processing requirements might justifiably expect flexibility in terms of modifying record formats and extending application programs. Although users of this class would probably consider the use of a DBMS as an unnecessary complexity, they are nevertheless still faced with the prospects of evolving data storage and application program requirements.

The study of four batch-processing users (see 2.6) indicated that the conventional facilities of the CRH system (i.e. tree-like record structure, sequential files with multiple record classes) were appropriate. Direct questioning about system changes and evolution indicated that format-independence would have been valuable for specific instances which were recalled. The fact that often main files were used by many programs supports this view since a format-independent system would minimise the effects of main file format changes. Of the existing systems which provide format-independence the most similar to the CRH system were NIMMS, IMS and the DBTG COBOL extensions. These systems are full procedural Data Base Management Systems and offer much more powerful general facilities and complex record storage-structures than the CRH system. However, the degree of format-independence offered by the CRH system is comparable. It is probably fair to say that the implementation of these other systems (which appeared chronologically after the CRH system was

implemented) is at least an order of magnitude more complex. During the design of the CRH system the "examination of existing software" (as required by the integrated approach to system design) mainly encompassed the Data Processing Languages since NIMMS, IMS and DBTG COBOL were not available. COBOL appeared the most flexible of these languages in terms of system evolution, since certain alterations to data could be accommodated by recompilation of the "procedure division". This does not provide a dynamic run-time linkage of program and data. Further, the data description is not stored with the data, record formats are still assumed by the processing programs.

The remaining requirement in addition to the consideration of user requirements and existing software, was for suitable hardware to support the facility of format-independence. Such hardware, already existed in the form of the "Basic Language Machine" described in the book Basic Machine Principles [42] and more specifically in a paper describing Basic Language[43]. Format-independence requires dynamic binding of program and data and this is conveniently achieved by addressing data indirectly via a system of pointers so that different record formats may be mapped by altering the pointer values without affecting compiled code - an operation which occurs only once when the file is opened. The Basic Language Machine lends itself to this type of implementation since, being a sequented machine, data is always addressed via descriptors called codewords which are interpreted by the hardware. Codewords can be used instead of pointers with added advantages in terms of speed and security over implementation on conventional machines.

The CRH system is primarily concerned with intra record structure and provides high-level record schema facilities for describing record classes using a fixed tree-like data structure. The CRH system record schemas differ from those in ECAL as indicated on page 87 . In contrast with the ECAL language inter-record relations are minimal being confined simply to user-determined

sequencing of records on file storage. During file processing the user is aware that data exists at two distinct levels. Data at the primary level or file area can be referenced directly while data at the secondary level or file store must be transferred to the primary level for processing. The file store consists of a collection of CRH files (henceforth simply called files). Each file is distinct and comprises a collection of records stored in some user-determined sequence. During file processing each file used by a program has a separate file area (in core-store). The logical unit of transfer between file and file area is the record, a file area can only hold a single record which is then termed the current record. Records can be transferred between storage levels one at a time under program control. A record schema provides a "window" through which the program views the file area and accesses record fields. Format-independence allows a program to view only part of the record in the file area and in this case the record schema declared in the program is in fact a sub-schema describing only the record fields of interest to the program. There are two basic modes of interaction with sequential file storage either to create new record instances by writing copies of successive current records from the file area to the file to create a completely new file or to retrieve record instances from an existing file by sequentially reading them into the file area. Format-independence is provided by the CRH system for "reading" information from an existing file. When "writing" information to create a completely new file record schema is always a complete description of the record class. Format-independence is not provided since this would imply an eventual synthesis of a record from a number of partial views. This was considered impractical without a formal framework for describing data-independence and further complicated by the sequential nature of the file storage medium. A composite mode of interaction with the file store involves modifying the contents of existing records, this is termed update. Using sequential files an "update" involves "reading" an existing file, modifying

existing record fields (in the file area) and "writing" the modified records to a new modified file. Format-independence is provided for "update" operation since the modified records will contain the fields not available to the program (i.e. not in the sub-schema) together with the fields which were altered (i.e. those in the sub-schema).

The language facilities of the CRH system (Data Description Language and Data Manipulation Language) are discussed in Section 4.2. Also, the format-independence facilities are explained and specific examples executed under the CRH system are shown. Section 4.3 briefly indicates the method of implementation.

Section 4.2 The CRH Language Facilities

4.2.1 Overview

4.2.2 The Data Description Language (DDL)

4.2.3 The Data Manipulation Language (DML)

4.2.4 Format Independence

4.2.5 Examples

4.2.1 Overview

The CRH system is made available to a user through two languages - a Data Description Language (DDL) and a Data Manipulation Language (DML). The DDL and DML augment the host Basic Language which provides the computational and procedural facilities. Programs consist of two parts, the Data Description written in DDL and the Program Body written in a mixture of DML and Basic Language. The DDL intrcduces four classes of object:- files, records, groups and items. The first three of these have no counterpart in Basic Language, but items have a correspondence with the basic data types of the host language (i.e. the Basic Language data types constitute record fields). The nature of the DML and its relation to the host language reflects these facts. The DML comprises a number of CRH Functions and a construct called a "Complete Item Name". The former manipulates files, records and groups while the latter serves as a reference to an item. In the Program Body which consists of a sequence of "lines", a CRH function separately occupies one complete line but complete item names can be intimately mixed with Basic Language on the same line and are valid in any context which permits a reference to a Basic Language data type.

Using the DDL a file schema is declared for each file used by a program, each file schema contains record schemas for those record classes known to the program. Depending on the type of file schema in which it is declared a record schema may either be interpreted as the record schema or a sub-schema of a record class. For example a write-file schema would contain "the record schemas" which dictate the actual contents and format of the file records, a read-file schema would contain "sub-schemas" which are sub-descriptions of the record schemas used in writing that file. Syntactically all record schemas are identical, it is their context which determines the way in which they are to be interpreted. Format-independence is provided by the record schema: sub-schema relationship which is established at run-time by the CRH system.

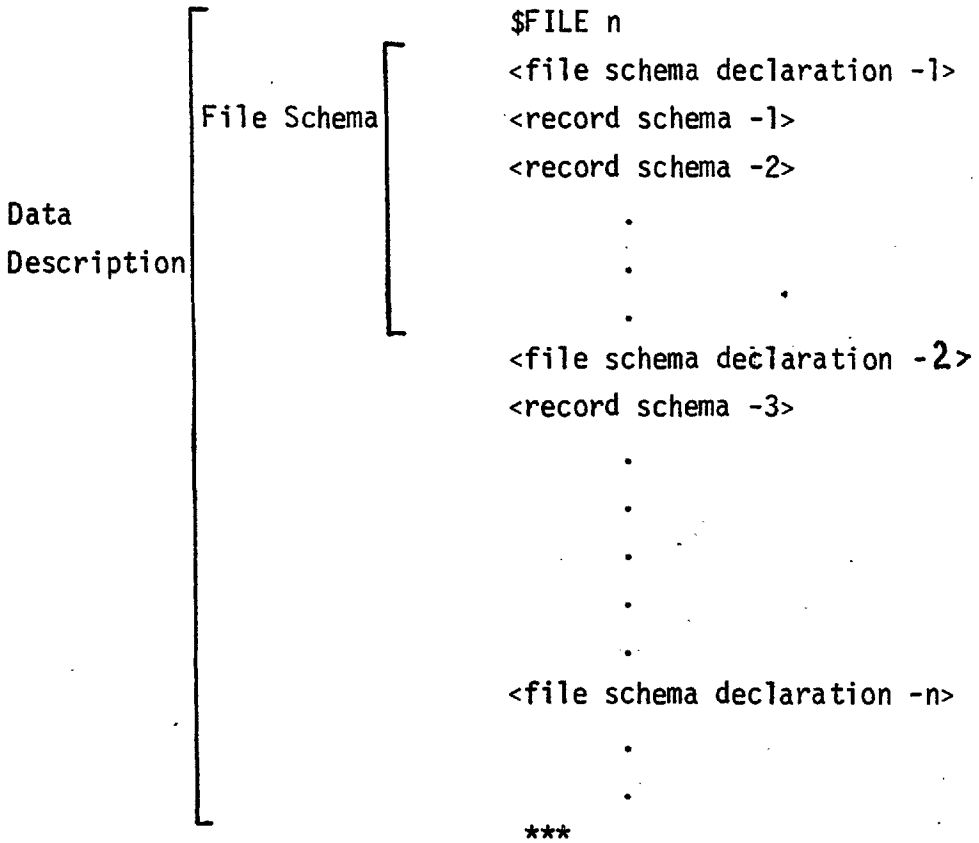
The language extensions are implemented by modifying the basic language assembler as indicated later in 4.3.2. These modifications require that the DDL and DML be distinguished from the host language and a "$" character serves this purpose. This also effects a separation between names declared in the program Data Description and those declared in the host language. Host language homonyms can be freely used without ambiguity except for the file schema names. In this case the host language homonym must identify the device on which the physical file is stored. Host language homonyms are discussed in Appendix E.21. There are certain features of the current implementation which are visible at the user level, these are concerned with record class identification and variable length record handling. A two byte record field is allocated for these purposes, and although this is never referenced by the user it must be declared as explained later under "Record Schemas" in 4.2.2 and in E.20. A similar situation occurs with the "TYPE declaration" of "Write-File Schemas" in 4.2.2 and described in E.19.

Error monitoring closely follows the pattern of the host language for simplicity of implementation. The types of error detected by the CRH system are listed in Appendix E.22.

Sub-section 4.2.2 provides an overview description of the DDL and DML. A more detailed description exists in Appendix E and is cross referenced where appropriate. Sub-section 4.2.3 discusses the extent of format-independence provided by the CRH system providing rules for validating the record schema: sub-schema relationship and illustrating these with examples. Sub-section 4.2.4 contains examples of file-processing programs which were assembled and executed successfully under the CRH system and make use of the format-independence it offers.

4.2.2 The Data Description Language

A program Data Description written in DDL has the general form:-



A file schema is declared for each CRH file to be used by the program. Each file schema contains a list of record schemas, one for each record class known to the program. The file schema declaration indicates how the file is to be used and hence how the record schemas are to be interpreted by the CRH system. There are four types of file schema:-

write-file schema: used for creating a new file

read-file schema: used for retrieving information from an existing file

update-file schema: used for creating a modified (updated) version of an existing file

core-file schema: used to provide workspace within a user program, is not associated with a physical file.

The syntax of the list of record schemas takes a similar form for each of the above types of file schema, it is the file schema declarations and the interpretation of the record schemas which vary.

Write-File Schemas (see Appendix E.2)

Example:
```
$FILA W
<record schema -1>
.
.
.
```

"FILA" is the file name whose scope is the current program and is used with the functions OPEN and CLOSE to allocate and free the physical file volume. The WRITE function transfers a current record to the physical file, READ functions are not permitted against this type of file. The list of record schemas define all the record classes which are on the file. Furthermore, they are the record schemas of those classes i.e. they describe all the record fields and determine the record format (see Fig. 4.12, "The Record Schema").

Read-File Schemas (see Appendix E.3)

Example:
```
$FILB R
TYPE n
<record schema -1>
.
.
.
<record schema -n>
```

"FILB" is the file name whose scope is the current program and is used with the functions OPEN, CLOSE and READ. The latter transfers the "next" physical record into the file area, and also indicates (by return of control) what class of record has been transferred (see READ function, E.14) "TYPE n" declares the number of record schemas which follow, this is an implementation feature of the current system (see Appendix E.19). The list of record schemas describes only those record classes of interest to the current program. Furthermore, they are interpreted as sub-schemas of those record classes i.e. they describe only those record fields required for use (see 4.2.4).

Update-File Schemas (see Appendix E.4)

```
Example:    $FILC W
            $FILD R
             TYPE n
            <record schema -1>
              .
              .
              .
            <record schema -n>
```

Physically a pair of files are involved for "reading" the original file and "writing" the modified file - this is reflected in the structure of the update file declaration. The OPEN function applied to "FILC" results in the allocation of both input and output volumes. The READ function can be applied to "FILD" to transfer a record into the file area where it can be modified and subsequently output using the WRITE function.

Core-File Schemas (see Appendix E.5)

```
Example:    $FILG C
            <record schema 1>
              .
              .
              .
              .
```

"FILG" is the file name used to OPEN the file. Core files are not associated with physical volumes, their file areas are used as temporary storage. READ and WRITE functions have no effect on core-files. The record schemas determine the complete mapping of the file area as for write-files.

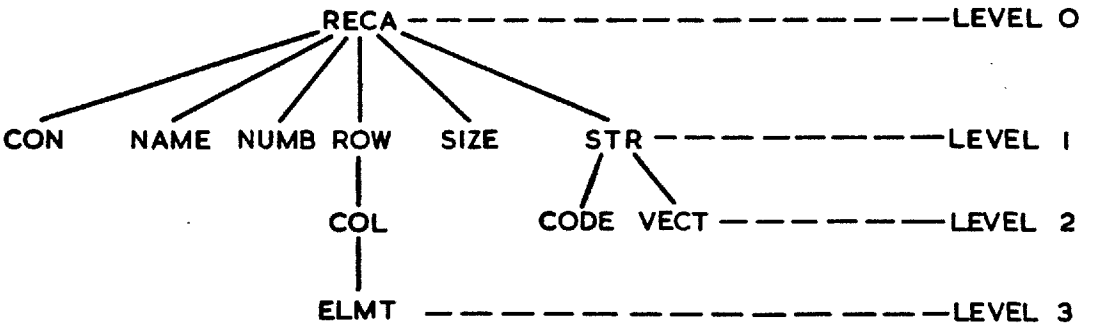
Record Schemas

Record schemas provide high level facilities for describing record classes. The record schema provides a tree-like data structure consisting of "groups" (see Appendix E.7) and "items" (see Appendix E.8). In Fig. 4.1 the groups are "ROW", "COL" and "STR". Although conceptually the user sees a tree structure this is linearised in storage and a record comprises a simple vector of fields (the mapping of the record vector into its constituent fields is termed the record format). Items of the record schema correspond to physical fields, which hold data values, the groups have no direct physical counterpart. Level numbers are used together with the sequencing of the group and item declarations to determine the tree structure as indicated in Fig. 4.1. It should be noted that the relative values of level numbers are important rather than their absolute values. For example although "ELMT" is conceptually at level 3 it need not have the value "3". This allows gaps to be left in the level numbering sequence to facilitate subsequent editing of record schemas.

Items correspond to the basic data types of the host language, for example in Fig. 4.1 "NUMB" is a single "word", "NAME" is a vector of eight bytes (i.e. the bytes are NAME..0.....NAME..7).

There are two types of group, simple and repeating groups "ROW" and "COL" are repeating groups, "STR" is a simple group. Simple groups allow a collection of items to be addressed by a single name for the purpose of copying via the GMV (Group-more) function. Repeating groups may also be used in the same way but additionally they indicate that record fields are repeated. For example the repeating group COL indicates that the field corresponding to ELMT is to be repeated 3 times, but COL is

TREE STRUCTURE



RECORD SCHEMA

```
RECA
I CON A1
I NAME  A2
I NUMB  N
I ROW  (3)
  2 COL(2)
    4  ELMT A3
I SIZE N
I STR
  2 CODE A2
  2 VECT N2
```

Fig. 4.1 Record Structure

itself a member of repeating group ROW so that the field is repeated 3 x 4 times. However at any one time the item ELMT can only be associated with one field - the function SEL (select) allows the particular field to be selected. The function SEL would have to be applied to both ROW and COL.

In the current implementation of the CRH system the user must declare a two byte item named "CON" after every record schema name. This item is never referenced by a user, it is used by the CRH system (see Appendix E.20). The last item in a record schema may be a variable item (see Appendix E.8) if desired and in this case the corresponding field length can vary from record to record. Examples containing a variable item occurs in 4.2.5 Programs 3 and 4.

4.2.3 The Data Manipulation Language

The Data Manipulation Language comprises a number of CRH Functions and a construct called a complete name. A complete name is an unambiguous reference to an object in the Data Description, complete names contain one or more components but in the simplest case they consist only of the object name. Where object names are duplicated in the Data Description the multi-component complete name allows unambiguous referencing as described in Appendix E.10. Complete item names are references to items in the data description, they may be embedded directly in the host language (an example is given in 4.2.5, Program 1) but other complete names can only occur as parameters to CRH Functions.

The CRH Functions of the DML have a similar form to the host language function calls in the interests of efficiency of transformation by the extended assembler. Parameter type-checking is performed at assembly-time.

There are eight CRH Functions which are described in detail in Appendix E.11-E.18. The function identifiers are listed below together with an indication of their usage:-

OPEN:	This allocates a physical volume and arranges the schema or sub-schema data mappings.
CLOSE:	This terminates the use of a file.
READ:	Transfers the next record from physical file to file area and identifies the record class.
WRITE:	Transfers the current record to the physical file.
GMV:	Moves the fields associated with a group.
SEL:	Selects a particular field occurrence in a repeating group.
VSP:	Adjusts the length of a variable field.
INIT:	Initialises the length of a variable length field of a current record.

4.2.4 Format-Independence

The CRH system provides format-independence by its record schema: sub-schema descriptive facilities. The record schemas of write-files are used to determine the physical record formats by a simple linearisation of the conceptual tree data-structure (see Fig. 4.12, The Record Schema). The record schemas of read and update-files must constitute a valid sub-description of the existing record classes. What constitutes a valid sub-description derives in part from logical constraints and in part from efficiency restrictions made in order to simplify the system implementation and maintain run-time efficiency. An example of the former is that a named item in the sub-schema must be present in the schema. An example of the latter is that similarly named items must correspond to similar fields (i.e. no type conversion). The relationship of simple groups in the schema and sub-schema is largely a matter of convention. On the assumption that a simple group is declared for the convenience of the application rather than being an intrinsic feature of the data then there is no relationship between simple groups in the schema and sub-schema i.e. simple groups may be ignored when comparing schema and sub-schema. Repeating groups however define a structuring of the record which effects a reduction in the number of items which have to be declared for collections of fields. Both the number of field repetitions and the way these are nested must correspond for items which belong to repeated groups.

There are five rules for establishing whether the schema: sub-schema relationship holds:-

Rule 1:

> The record sub-schema name must be the same as the record schema name.

Rule 2:

> For each item in the sub-schema there must be a corresponding item with the same name and attributes in the schema.

Rule 3:

For each repeating group occurring in the sub-schema there must be a corresponding repeating group in the schema with the same name and attributes (i.e. the number of repetitions must be the same).

Rule 4:

If an item satisfying rule 2 is a member of n repeating groups in the sub-schema then that item in the schema must be a member of n and only n corresponding repeating groups nested in the same way. (Note:- this does not imply that the level numbers correspond in value).

Rule 5:

This is the inverse of rule 4 generated by interchanging "schema" and "sub-schema".

Record Sub-Schema Examples

Examples (i) - (x) below are all valid record schemas. If these examples are compared with the record schemas of corresponding names using the rules for deriving sub-schemas, then examples (i) - (v) are valid sub-schemas. However (vi) - (x) are invalid sub-schemas for the following reasons:-

(vi) the field VECT has different attributes (contravenes rule 2).

(vii) ROW and COL are in different order. (contravences rule 4).

(viii) LINE does not belong to x repeating group in the sub-schema (con-ravene rule 5).

(ix) FIR is not present in the record schema (contravene rule 3).

(x) NUMB does not belong to a repeating group in the record schema (contravene rule 4).

Valid Record Sub-Schemas

(i)
```
RECA
2 CON A1
2 VECT N2
2 NUMB N
2 ROW (3)
  4 COL (2)
    5 ELMT A3
```

(ii)
```
RECA
1 CON A1
1 GRP
  2 CODE A2
  2 NAME A7
  2 NUMB N
```

(iii)
```
RECB
2 CON A1
2 PTR (1)
  3 COL N
  3 ROW N
1 VAR V15
```

(iv)
```
RECB
1 CON A1
1 PTR (1)
  5 COL N
1 NUMB N
```

The Record Schemas

```
RECA
1 CON A1
1 NAME A7
1 NUMB N
1 ROW (3)
  2 COL (2)
    4 ELMT A3
1 SIZE N
1 STR
  2 CODE A2
  2 VECT N2
```

```
RECB
1 CON A1
1 NUMB N
1 PTR (1)
  5 ROW N
  5 COL N
1 NAME A7
1 ADDR (2)
  2 LINE A9
1 VAR V15
```

Valid Record Sub-Schemas

(v) RECC
1 CON A1
1 GRP (3)
2 LOC A3
2 ATOT N

The Record Schemas

RECC
1 CON A1
1 GRP (3)
2 ATOT N
2 SIMP
3 RG (2)
4 NAME A3
3 LOC A3
1 VALS N1

Invalid Record Sub-Schemas

(vi) RECA
1 CON A1
1 VECT N4
1 NUMB N

(vii) RECA
1 CON A1
1 COL (2)
2 ROW (3)
3 ELMT A3

(viii) RECB
4 CON A1
4 NAME A7
4 LINE A9

(ix) RECB
1 CON A1
1 NUMB N
1 FIR (2)
2 NAME A7

(x) RECB
1 CON A1
1 ADDR (2)
4 LINE A9
4 NUMB N

4.2.5 Examples

Four example programs were written to demonstrate the format-independence described earlier. These programs were compiled and executed under the current version of the CRH system.

> Program 1 produced a file with two record classes. This file was read by Program 2 which only processed one class and provided a record sub-schema for this class.
>
> Program 3 produced a file with one variable length record class which was processed by Program 4 using a sub-schema.

The record schemas and sub-schemas used were selected from those appearing in 4.3.2.

Since a description of the host Basic Language has not been provided, the basic language statements have been removed from the program listings which appear later - they have been replaced by english statements indicating the functions of the code removed. The interested reader may refer to Appendix F which contains the complete program listing of which Fig. 4.2 is part.

Examples of the use of DML statements are provided in the program descriptions which follow.

Program 1

Referring to Fig. 4.2 the Data Description of Program 1 contains a single write-file schema "FILB" which has two record schemas "RECA" and "RECB". The program body contains one main loop which outputs a record of class "RECA" followed by a record of class "RECB". The file is opened prior to use and closed after four iterations of the loop. Following the execution program 1 the data records of the created file were listed by printing as a character string using a standard ICL program utility. Fig. 4.3 contains the listing which has been reorganised so that records of the same class are contiguous for ease of presentation. (The records are interleaved on physical storage as indicated by

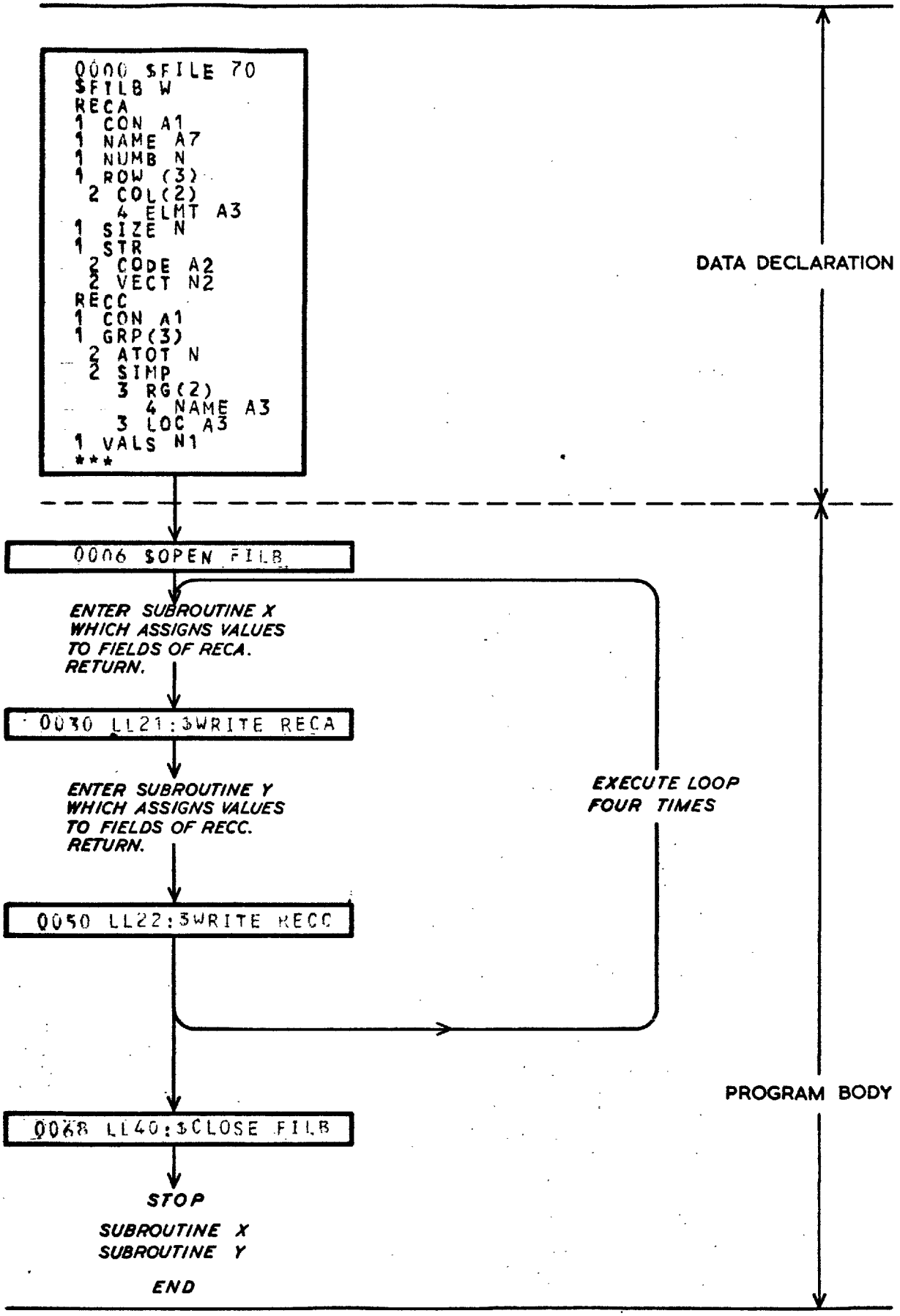


Fig. 4.2 Modified Listing of Program 1

RECORD CLASS "RECA"

				ROW 0			ROW 1			ROW 2			ROW 3					
CONT	NAME	NUMB	COL,0 ELMT	COL,1	COL,2	COL,0	COL,1	COL,2	COL,0	COL,1	COL,2	COL,0	COL,1	COL,2	SIZE	CODE	VECT
1#00	RECA	NO1	0009	E100	E101	E102	E110	E111	E112	E120	E121	E122	E130	E131	E132	0005	BA10	000100020003
1#00	RECA	NO2	000:	E200	E201	E202	E210	E211	E212	E220	E221	E222	E230	E231	E232	0006	BA20	000200030004
1#00	RECA	NO3	000;	E300	E301	E302	E310	E311	E312	E320	E321	E322	E330	E331	E332	0007	BA30	000300040005
1#00	RECA	NO4	000<	E400	E401	E402	E410	E411	E412	E420	E421	E422	E430	E431	E432	0008	BA40	000400050006

RECORD CLASS "REC B"

		GRP.0				GRP.1				GRP.2				GRP.3				
CONT	ATOT	RG.0 NAME	RG.1	RG.2	LOC	RG.0	RG.1	RG.2		RG.0	RG.1	RG.2		RG.0	RG.1	RG.2		VALS
2+00	0000	N100	N101	N102	LC10	0001	N110	N111	N112	LC11	0002	N120	N121	N122	LC12	0003	N130	N131	N132	LC13	00020003
2+00	0000	N200	N201	N202	LC20	0001	N210	N211	N212	LC21	0002	N220	N221	N222	LC22	0003	N230	N231	N232	LC23	00030004
2+00	0000	N300	N301	N302	LC30	0001	N310	N311	N312	LC31	0002	N320	N321	N322	LC32	0003	N330	N331	N332	LC33	00040005
2+00	0000	N400	N401	N402	LC40	0001	N410	N411	N412	LC41	0002	N420	N421	N422	LC42	0003	N430	N431	N432	LC43	00050006

Fig. 4.3 Listing of File Produced by Program 1

```
    0144 X4 LD 3
    0145 X2 LD #25
    0147 % HERE THE VALUES OF "ELMT" ARE INITIALISED, THE FUNCTION SEL
    0147 % MUST BE USED ON BOTH "ROW" AND "COL"
A→  0147 ROW:$SEL ROW,X4
    0163 X3 LD 2
B→  0164 COL:$SEL COL,X3
C→  0179 X0 LD $ELMT!
    0182 LL3:X0 MV X2
    0186 X0 MOD 1
    0187 X0 MV X7
    0188 X0 MOD 1
    0189 X0 MV X4
    0190 X0 MOD 1
    0191 X0 MV X3
    0192
    0192 X3 JNL COL
    0193 X4 JNL ROW
    0194  J X1
```

Fig. 4.4 Extract from Subroutine X of Program 1

the program logic). The record format is a simple linearisation of the record schemas with all fields word-aligned. The repeating group structure results in many fields for one item, in "RECA" the item "ELMT" has twelve corresponding fields. However, at any one time it is associated with one field and in the listing is shown associated with the first field occurrence.

The use of the function SEL and a "complete item name" is illustrated in Fig. 4.4 which is an extract from subroutine X of program 1. The host Basic Language statements always have a register name (X0, X1....X7) as argument. The line marked A is a CRH function call preceded by a label "ROW". The arguments of the SEL function indicate that the repeating group ROW is to be indexed by the contents of X4 which contains a small integer. The line marked B shows the SEL function applied to COL. The two SEL functions are embedded in two nested loops which index each element of COL in turn within ROW. Line C shows a "complete item name" embedded in a basic language statement which has the effect of loading into X0 the address (codeword) of the field currently associated with "ELMT" so that a value may be assigned. The overall effect of this code extract is to assign values to all the repeated fields.

Program 2

Program 2 illustrated in Fig. 4.5 reads the file produced by Program 1. The Data Description contains a read-file schema which has only one record schema "RECA". Furthermore this is a sub-schema of the RECA record schema used in Program 1. The OPEN function causes the schema: sub-schema relationship to be checked according to the rules of 4.2.4 and an appropriate data mapping arranged (see Fig. 4.12, A Sub-Schema). The READ function has three possible returns of control in this case (in general n + 2 where n is the number of record schemas declared). Control is returned directly to the instruction after the function call if the record type is not recognised. Otherwise control is transferred to one of a vector of labels passed as a parameter to the READ function."SV" is a vector of two labels

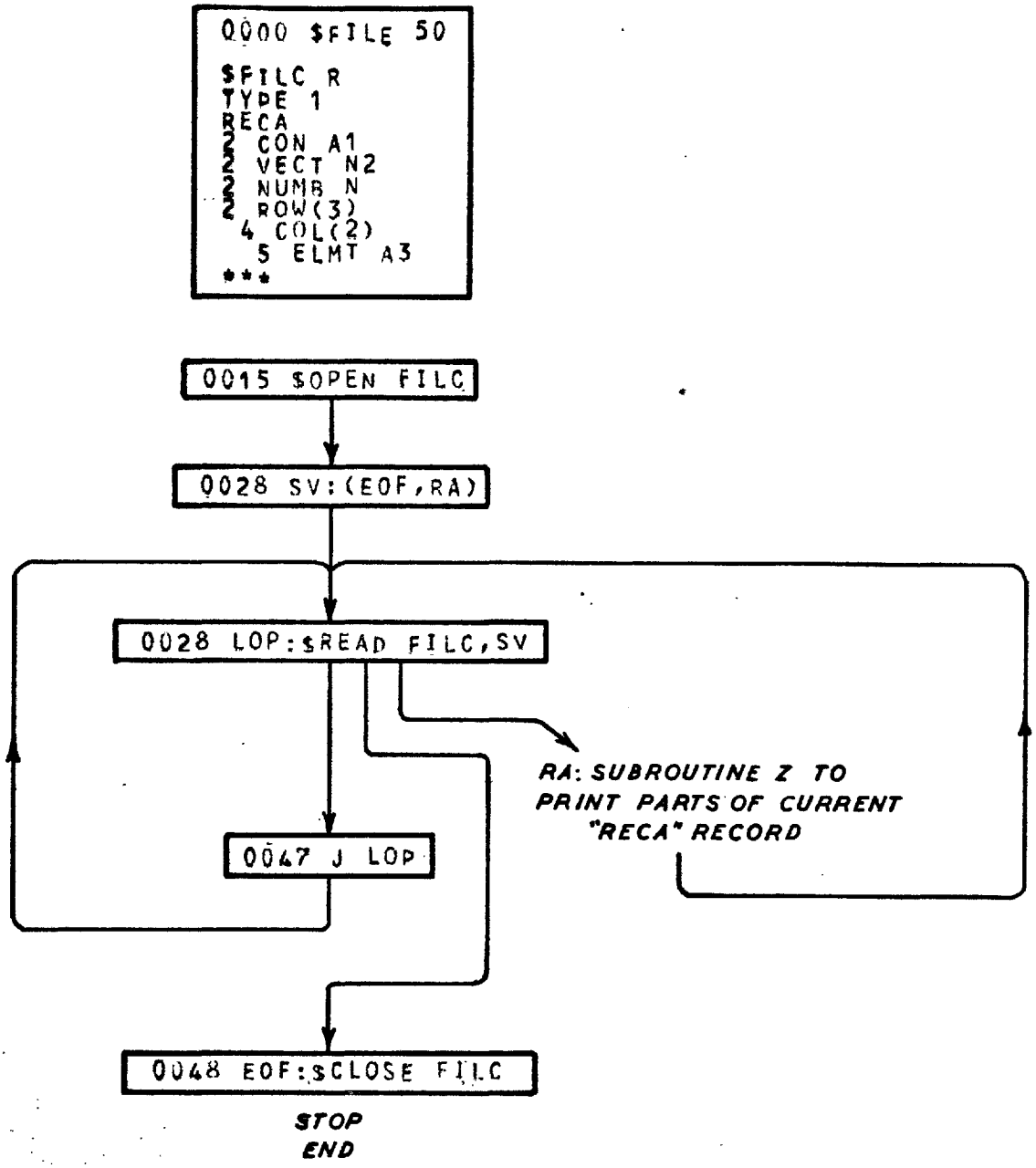


Fig. 4.5 Modified Listing of Program 2

```
   RECORD CLASS RECA
COLUMN 0     E100
COLUMN 0     E110
COLUMN 0     E120
COLUMN 0     E130
NUMB         0009
VECT.0       0001
VECT.1       0002
VECT.2       0003
   RECORD CLASS RECA
COLUMN 0     E200
COLUMN 0     E210
COLUMN 0     E220
COLUMN 0     E230
NUMB         0008
VECT.0       0002
VECT.1       0003
VECT.2       0004
   RECORD CLASS RECA
COLUMN 0     E300
COLUMN 0     E310
COLUMN 0     E320
COLUMN 0     E330
NUMB         0007
VECT.0       0003
VECT.1       0004
VECT.2       0005
   RECORD CLASS RECA
COLUMN 0     E400
COLUMN 0     E410
COLUMN 0     E420
COLUMN 0     E430
NUMB         0006
VECT.0       0004
VECT.1       0005
VECT.2       0006
```

Fig. 4.6 Listing of Output from Program 2

"EOF" and "RA". Control is transferred to the first label (EOF) at the end of file condition or to the second label (RA) if the record just read belongs to the RECA class. The subroutine "Z" prints the values of certain record fields as seen through the sub-schema. Fig. 4.6 gives the printed output of Program 2, the field values have been preceded by a literal indicating how they appeared to the program via the sub-schema. These field values compare with the actual field values present in the records indicated in Fig. 4.3. The CRH system has therefore arranged that the sub-schema provides the appropriate view of the data.

Program 3

Program 3 illustrated in Fig. 4.7 creates a file with one class of variable length records. The last field "VAR" in the record schema may vary in length from one to sixteen bytes. The program has a single loop which is iterated four times. Subroutine U assigns values to the fields of the current record and since one of the record fields is variable the length of this field must be adjusted. An example of the VSP function appears in sub-routine U. The two parameters indicate the item to which the function is applied and the required length of the variable field (in register X0). A listing of the records produced by Program 3 appears in Fig. 4.8. The variable field is last and since I/O transfers involve an integral number of words there are some zero packing bytes.

Program 4 illustrated in Fig. 4.9 reads the file created by Program 3. The READ function distinguishes the record class required. It should be noted that the loop to ignore unrecognised records is present because the program must not make the assumption that only one record class exists. Following the successfully recognition of a RECB record and prior to printing the function INIT is used. This function adjusts the codeword corresponding to VAR so that the correct length of field is addressed (i.e. ignoring packing bytes). The record schema name parameter of INIT uniquely identifies the variable item since there can only be one variable field in a record. The resulting print of the record fields can be found in Fig. 4.10. This may be compared with the record contents of Fig. 4.8.

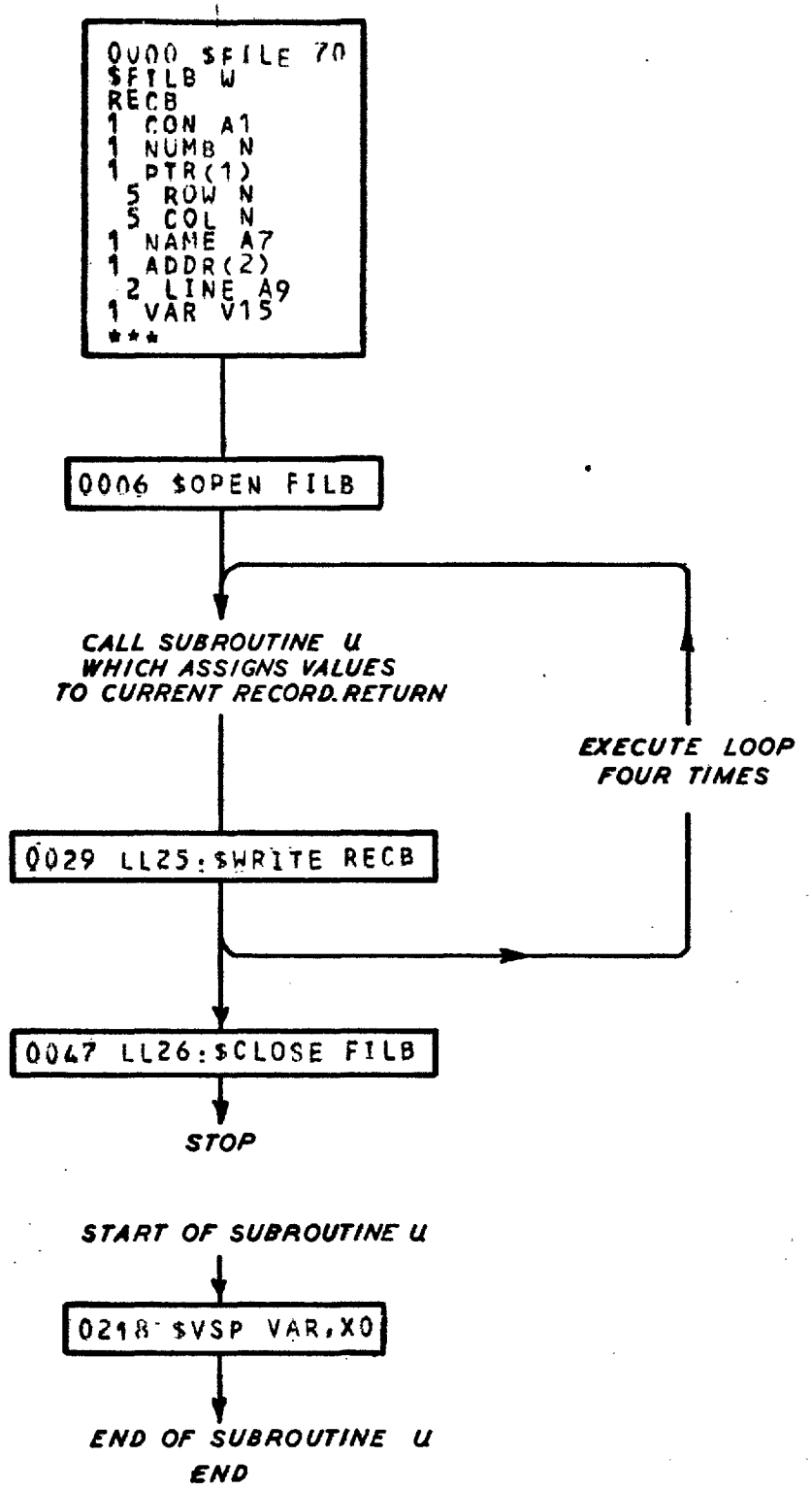


Fig. 4.7 Modified Listing of Program 3

CONT	NUMB	PTR.O ROW	PTR.O COL	PTR 1		NAME	ADDR. O LINE	ADDR. 1	ADDR. 2	VAR
1<00	0001	0001	0002	0001	0001	DEALER 1	STREET NO100	AREA NO100	TOWN NO100	VARIABLEA000
1=00	0004	0001	0002	0001	0001	DEALER 2	STREET NO200	AREA NO200	TOWN NO200	VARIABLEAA00
1>00	0005	0001	0002	0001	0001	DEALER 3	STREET NO300	AREA NO300	TOWN NO300	VARIABLEAAA0
1?00	0004	0001	0002	0001	0001	DEALER 4	STREET NO400	AREA NO400	TOWN NO400	VARIABLEAAAA

Fig. 4.8 Listing of File Produced by Program 3

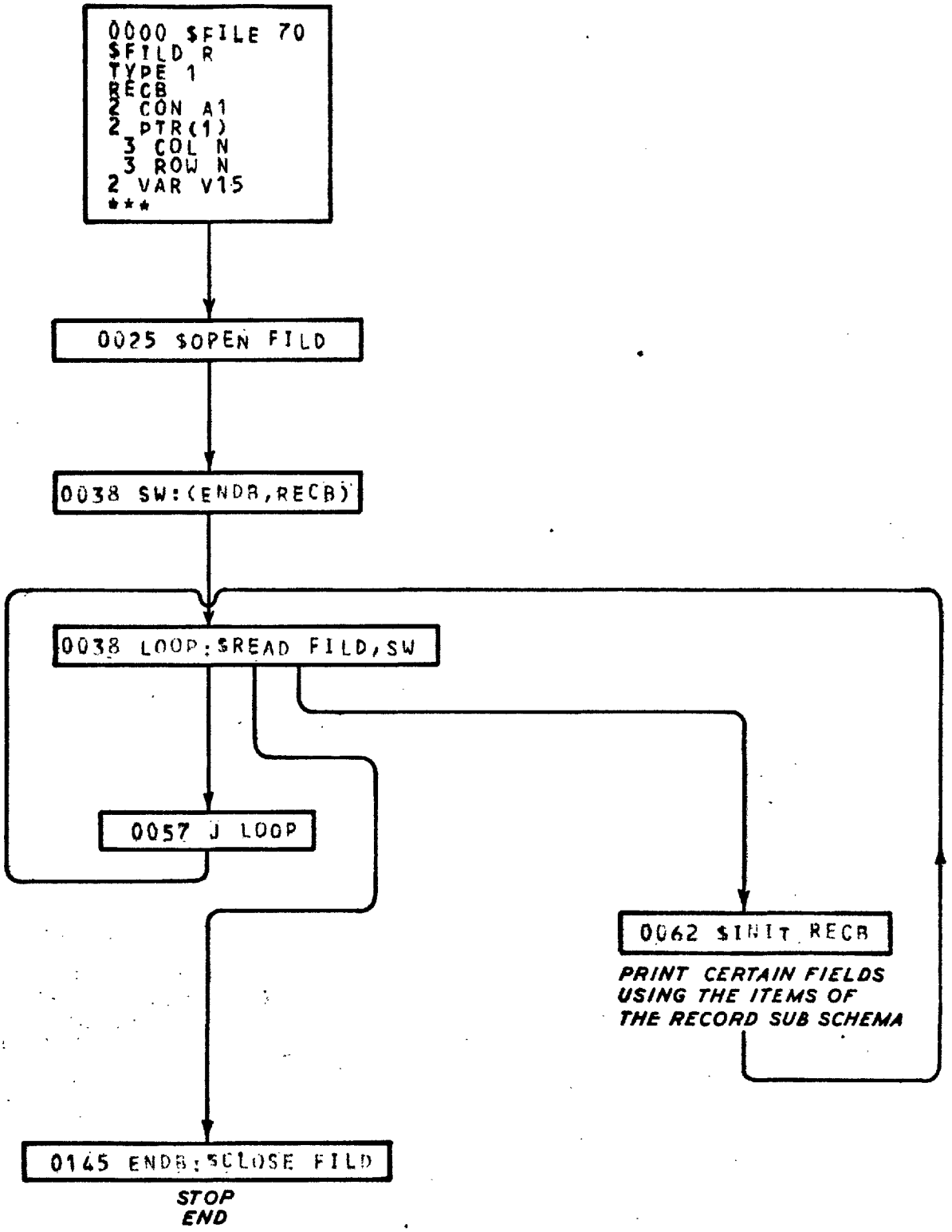


Fig. 4.9 Modified Listing of Program 4

```
ROW 0          0001
COL 0          0002
ROW 1          0001
COL 1          0001
VAR            VARIABLEA
ROW 0          0001
COL 0          0002
ROW 1          0001
COL 1          0001
VAR            VARIABLEAA
ROW 0          0001
COL 0          0002
ROW 1          0001
COL 1          0001
VAR            VARIABLEAAA
ROW 0          0001
COL 0          0002
ROW 1          0001
COL 1          0001
VAR            VARIABLEAAAA
```

Fig. 4.10 Listing of Output from Program 4

Section 4.3 Implementation of the CRH System

4.3.1 Overview

4.3.2 Assembler Extensions

4.3.3 The BLM/Relative Codewords

4.3.4 The Data Reference Structure (DRS)

4.3.5 File Headers

4.3.6 Mapping of Data and Record Areas

4.3.7 The OPEN Function

4.3.1 Overview

This section provides a brief description of the implementation of the CRH system, the most important points are covered though no attempt is made to provide an exhaustive description.

The CRH system requires two classes of software, compile-time software to support the DDL and DML and run-time software which actually performs the data manipulations directed by the DML. Both the DDL and DML are language extensions to the host Basic Language. In implementing language extensions the choices are generally either to use a pre-processor approach or to modify the language compiler. In the case of the CRH extensions the pure pre-processor method was not possible, primarily because the codeword structures involved in the file and record area mappings (see 4.3.6) could not be generated by using Basic Language alone. Compiler modifications (in this case assembler extensions) can be problematic in themselves introducing difficulties over and above those present in the actual language extensions. Consequently, an interface was chosen with the Basic Language assembler with the intention of minimising assembler modifications and to use where possible simple pre-processing techniques (which were primarily applicable to the DML extensions). The structural modularity of the Basic Language assembler greatly facilitated the identification of a suitable interface as described later. The run-time software presented less problems in terms of integration into the Basic Language operating system. Using standard facilities for defining new "library" procedures a new procedure was defined for each of the CRH functions.

The run-time binding of code and data when a file is opened requires both program and file data descriptions to be present at run-time. Furthermore, arrangements must be made for remapping the data in accordance with the two data descriptions. Two main alternatives present themselves, either data is addressed indirectly or the record itself is physically copied and transformed during this copy operation. The latter course

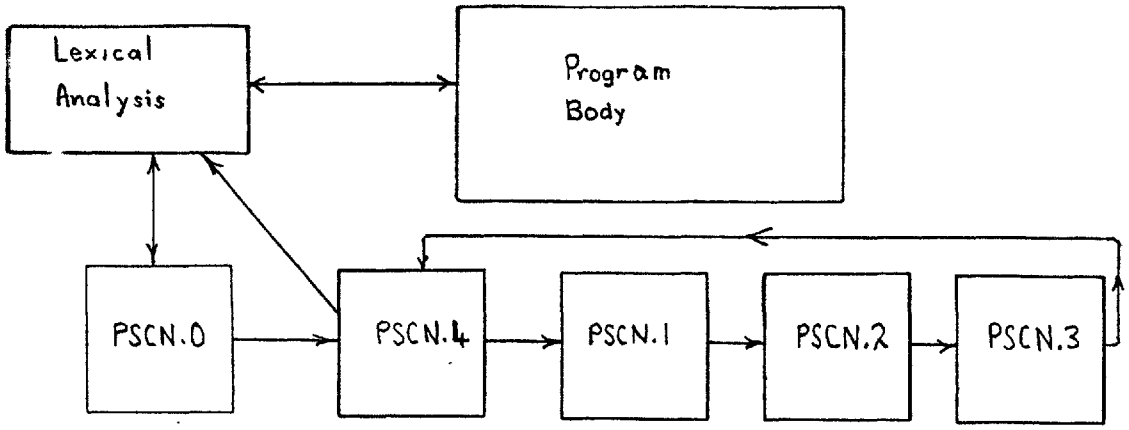
of action might be economically viable for highly variable records (as in DBTG COBOL) but indirect addressing is more practical for the fixed records of the CRH system. The relative codewords of the BLM provide a more efficient and secure means of indirect addressing than on a conventional machine since they are hardware interpreted with bound checking. The basic requirements for run-time binding are met by the Data Reference Structure (DRS) and File Headers. The DRS is stored in the object program and contains the complete program data description in a coded form together with the codewords used for indirect addressing. File headers are stored in the file and include a description of each record class on the file, i.e. the file data description. The OPEN function accesses both the file and program data descriptions and adjusts the codewords to accommodate the mapping of the physical records. Because CRH records are fixed format no further alteration of the mapping is required during subsequent file processing. Although the structures generated by the OPEN function to map records could not be produced using pure Basic Language this does not lead to complications in store management since these structures are contained within a single machine segment.

The Basic Language Assembler modifications are described in 4.3.2, Relative Codewords are discussed in 4.3.3. The DRS and File Headers are outlined in 4.3.4 and 4.3.5. The method of mapping data areas via relative codewords is illustrated in 4.3.6 and finally the OPEN function which performs the remapping is described in 4.3.7. The other CRH functions are not described in this section since they are of secondary importance to the OPEN function which provides the format-independence.

4.3.2 Basic Language Assembler Extensions

The standard Basic Language Assembler consists of a number of related code segments, but for purposes of this thesis can be considered as composed of two main parts, the Lexical Analysis Segment and the Assembler Body. Programs are assembled line-by-line. Each input line is transformed by lexical analysis into a more compact and convenient form called a "token representation" i.e. identifiers are extracted and placed in a list of fixed size elements and the line is compressed by removing blanks and replacing operator symbols with appropriate codes. The transformer line is then passed to the assembler body where further checks are performed and code is generated, various tables are used for forward jumps etc. and for connecting references to relative codewords declared within the program. A local Name Table contains symbolic identifiers of declared relative codewords and information to enable references to be directed to these codewords.

As mentioned in 4.3.1 the structures used in the CRH system could not be generated by pure Basic Language. An interface was chosen with the standard assembler which allowed access to the Local Name Table and to the assembly area so that codewords could be created in the latter (as required by the DRS) and corresponding entries made in the former. The interface is effectively between the lexical analysis segment and the program body, after the symbolic text has been transformed (although entry and exit to the CRH assembler routines is still made to the lexical analysis segment for convenience). When CRH codewords are created the corresponding Local Name Table entries are made to contain names which are invalid in the symbolic text. However, since name validity checks are performed only in the lexical analysis segment the CRH assembly routines can insert these invalid names in the "token representation" to reference the CRH codewords. The effect of this is to prevent unauthorised reference to CRH codewords which might cause system corruption and failure.



Extended Basic Language Assembler

The structure of the extended assembler is shown above, the CRH assembler extensions are the segments PSCN.0 through PSCN.4. Each of these processes a particular CRH language form:-

PSCN.0 - Complete Item Names embedded in basic language text

PSCN.4 - CRH functions

PSCN.1.3- Data Description

Entry into PSCN.0 is always triggered by the "$" character. A minor modification to the lexical analysis segment allows the $ to be regarded as an operator and arranges for the entry to PSCN.0 following the creation of the token representation. PSCN.0 decides whether a CRH function or complete item name is present. If the former then it calls PSCN.4, otherwise it identifies the corresponding CRH codeword and replaces the complete item name with the (invalid) name of the codeword then returns to the Lexical Analysis Segment. PSCN.4 determines whether the CRH function was $FILE (indicating the Data Description), if so it calls PSCN.1. Otherwise it processes the CRH function call, checking that the parameter types are correct and effectively performs a pre-processing operation inserting additional parameters as required then returns to the Lexical analysis segment. PSCN.1 takes control of the input device and reads the subsequent lines of the Data Description directly, performing syntax checking purely within each line and creating the "Coded

Data Description" of the DRS (see 4.3.4) together with space for the Codeword Mapping List and Data Area. PSCN.2 and 3 then generate the Codeword Mapping List by processing the Coded Data Description, checking that it is of the correct overall form and that the record schemas represent valid tree structures (i.e. nodes of the tree are groups and terminals are items). Following successful creation of the DRS control is returned to PSCN.4 and then to the Lexical Analysis Segment.

4.3.3 The Basic Language Machine/Relative Codewords

A characteristic of the Basic Language Machine (BLM) is that "addresses" take the form of codewords not, as in conventional machines, a simple integer (possibly modified dynamically by a "base" value in a register). A codeword contains identification of the "type" of information addressed, and an "index" in addition to an integer corresponding to a conventional machine address. Both the "type" and "index" are interpreted by the hardware, as is the integer address. The codeword is analogous to the "descriptor" of the Burroughs B6500 machine [44] . However, there is a fundamental difference in philosophy between the two machines. The descriptor is intended as an aid to the implementation of an Algol-Like assembly language in the B6500, users can be unaware of the existence of descriptors. In Basic Language (the assembly language of the BLM), the codeword plays a central role. The language provides secure facilities for creating and manipulating codewords and the vectors they reference, actual codeword formats are concealed from the user.

There are two main classes of codeword relative and absolute. Absolute codewords reference "segments" of store while relative codewords reference vectors of store within the segments in which they are located. Absolute codewords will not be discussed in detail, relative codewords are fundamental to the implementation of the CRH system.

Relative codewords occupy a single "word" of storage (32 bits) and comprise the following parts recognised by the machine hardware:-

relative codeword

The "type" indicates the class of vector referenced, four values of type are relevant to the implementation of the CRH system:-

(i) Word Vectors

(ii) Byte Vectors

(iii) Relative Codeword Vectors

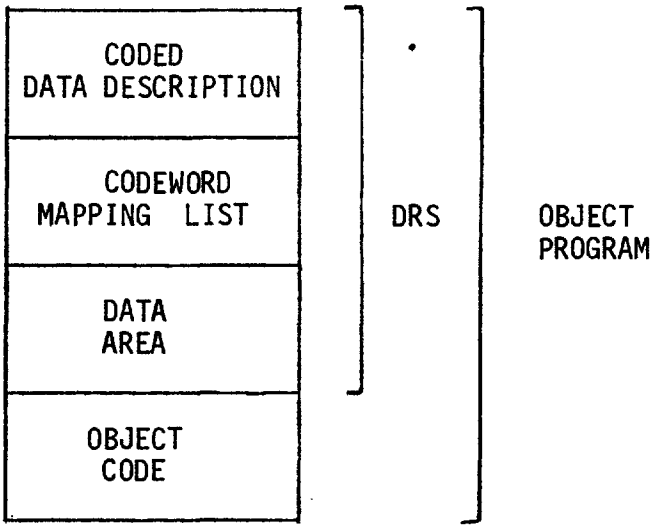
(iv) Undefined Codeword.

The Index gives the number of elements in the vector as follows:- number of elements = Index + 1. The n elements of the set are identified by the numbers 0, 1...(n-1). The relative address gives the position of the first element of the vector relative to the codeword, in units depending on the "type" (i.e. a relative byte address for byte vectors, relative word address for word and codeword vectors). Both the relative codeword and the referenced vector will lie in the same segment.

For an undefined codeword the index and relative address fields are not interpreted.

4.3.4 The Data Reference Structure (DRS)

The DRS is produced in the object code as a result of compiling the program Data Description. The DRS is available to all CRH functions at run time although it is primarily used and extensively modified at run-time by the OPEN function. The different parts of an object program using the CRH facilities are indicated below:-



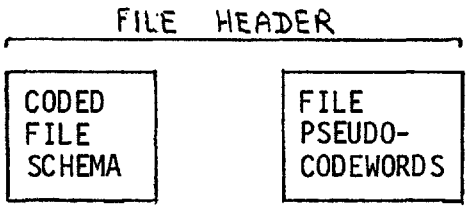
The Coded Data Description contains all the information present in the program Data Description in a more compact coded form. It consists of a simple list of variable length byte elements. Each element is derived from the corresponding object declaration in the Data Description and contains the symbolic name for the object together with appropriate codes for other characteristics. The Codeword Mapping List consists of a vector of relative codewords in one-one correspondence with the elements of the coded Data Description. References to CRH objects in the symbolic program have been converted during compilation to references from the object program to the appropriate codewords of the Codeword Mapping List. Before the object program is executed these codewords have the format "undefined" (see 4.3.3)

The Data Area will be used to allocate File Areas at run-time when the different files are opened. Following compilation the data area is null. The allocation of file areas and conversion of undefined codewords into valid codewords referencing the data areas is performed at run-time by the OPEN function (see 4.3.7). This allows alternative mappings of the file areas according to the record formats. The mapping of the data area is described in 4.3.6.

Although all the codewords have the "undefined" format after compilation the codewords belonging to write and core-file schemas are called pseudo-codewords and differ from those belonging to read and update file schemas. Pseudo-codewords contain in a compressed form type and size information and the "address field" contains the relative byte address of the corresponding field in the record. The conversion of pseudo-codewords into the corresponding valid codewords is a rapid operation requiring only the address of the file area. Pseudo-codewords can be generated directly in the case of write and core-files since their record formats are determined at compile time. Pseudo-codewords cannot be generated for the other types of file because the record mapping is established at run-time.

4.3.5 File Headers

Each CRH file has a file header which is not made available to a user program. The file header consists of two blocks as indicated below:-



The Coded File Schema is extracted from the DRS (see 4.3.4) of the program used to write the file. It comprises mainly the coded form of the record schemas describing the file classes. The File Pseudo-Codewords are extracted from the same DRS and provide the detailed record formats.

4.3.6 Mapping of the Data and Record Areas

There are two levels of mapping, firstly the overall mapping of the DRS Data Area into file areas and record areas, and secondly the mapping of record areas into constituent fields. Fig. 4.11 indicates how the data area would be mapped for two hypothetical file schemas, a write-file and a read-file respectively. File areas are allocated when files are opened so the diagram indicates that "FILA" was opened before "FILB". Record areas overlap since only one file record can be present at any one time. The file area for FILB has a vector called "type bytes" which is used by the READ function to identify the different record classes (see 4.2.5, Program 2).

The run-time mapping of the record areas into fields by the relative codewords gives the format-independence facility of the CRH system. Fig. 4.12 illustrates how the codewords are used to map a record in two cases. Firstly the record schema (used in program 1, fig. 4.2) gives rise to a corresponding vector of codewords in the Codeword Mapping List drawn directly underneath. The physical record (centre) is formatted in direct correspondence and completely mapped by these codewords. The record sub-schema (used in Program 2, Fig. 4.5) gives rise to a corresponding codeword vector in its codeword mapping list. The fields of the physical record relevant to this view of the record are referenced by these codewords. Not all the fields are referenced and there is no relationship between the sequencing of record fields and codewords.

Data Description

```
$ FILE 500
$ FILA W
  RECA
    |
  RECB
    |
$ FILB R
  TYPE 3
  REC 2
    |
  REC 2
    |
  REC 3
    |
  * * *
```

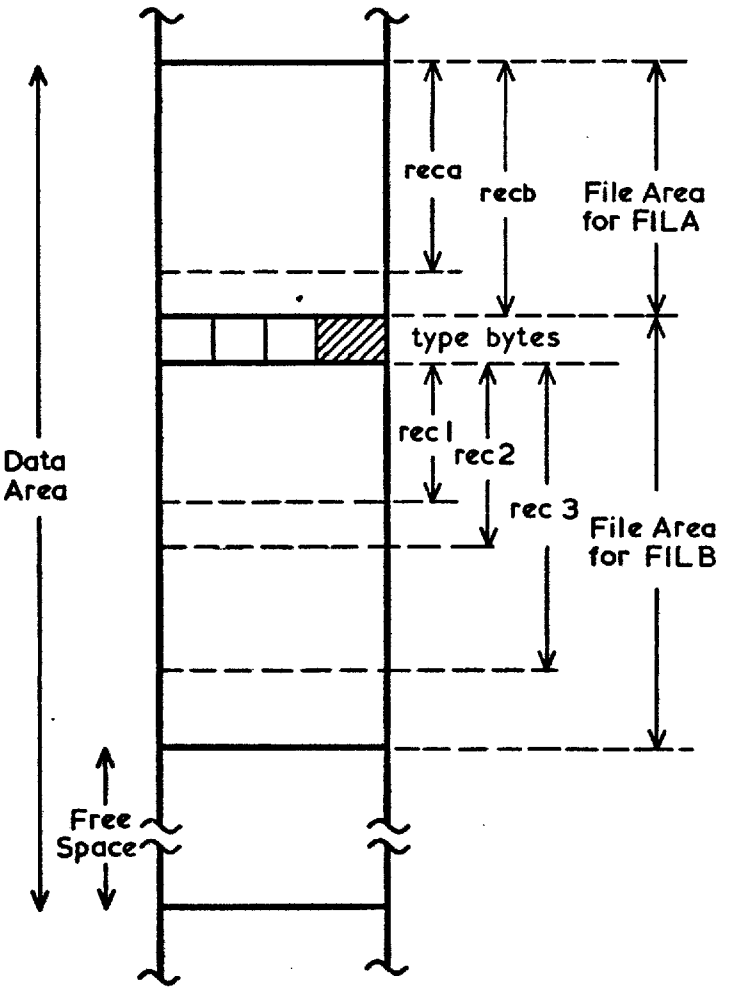


Fig. 4.11 Mapping of Data Area

The Record Schema

```
RECA
1  CON  A1
1  NAME A7
1  NUMB N
1  ROW (3)
   2 COL (2)
     4 ELMT A3
1  SIZE N
1  STR
   2 CODE A2
   2 VECT N2
```

A Sub-Schema

```
RECA
2 CON  A1
2 VECT N2
2 NUMB N
2 ROW (3)
   4 COL (2)
   5 ELMT A3
```

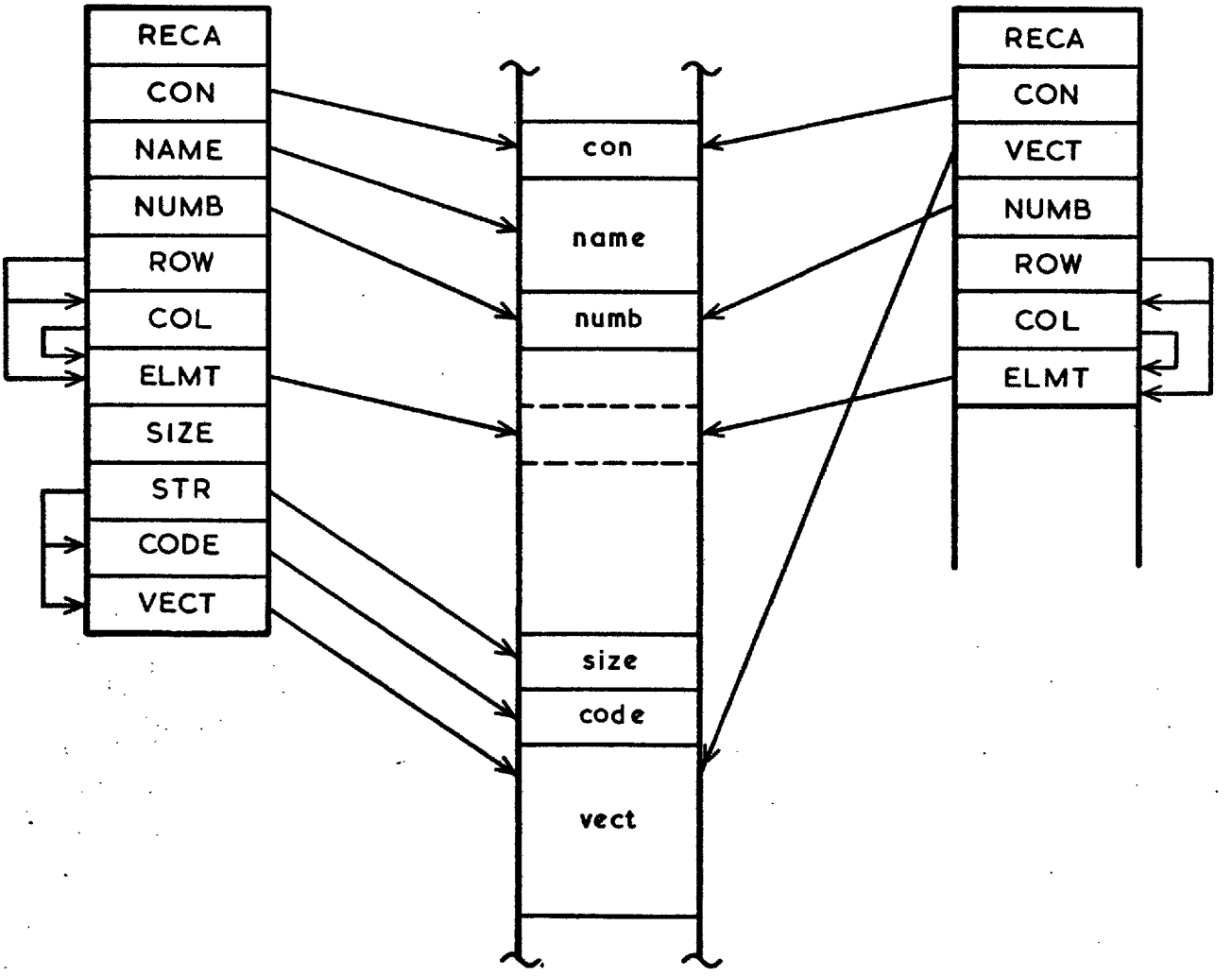


Fig. 4.12 Mapping of Record Areas

4.3.7 The OPEN Function

The OPEN function is the most complex of the CRH functions, it provides the format-independence of the CRH system. The OPEN function operates differently on each of the different types of file, the two basic operations being opening write-files and opening read-files.

Opening a write-file involves the following sequence of steps:-

(i) The DRS is accessed and the parts of the Coded Data Description and Codeword Mapping List corresponding to the file are identified. These will be called the coded file schema and the file pseudo-codewords in what follows.

(ii) The coded file schema and file pseudo-codewords are output as the first two blocks on the file, constituting the file header (see 4.3.5).

(iii) Various output buffers for the physical file handling are allocated external to the program.

(iv) The file area is allocated in the DRS Data Area (see 4.3.4).

(v) The file pseudo-codeword list is now transformed into valid codewords which map the record areas into fields. This gives rise to the sort of structure illustrated in Fig. 4.12, "The Record Schema".

Opening a Read-file involves the following sequence of steps:-

(i) The Coded File Schema (Sp) and the corresponding list of undefined codewords (Cp) are identified in the DRS.

(ii) The File Header containing the Coded File Schema (Sf) and the File Pseudo Codeword (Cf) are read. These give a complete description of the record format including symbolic identifiers.

(iii) Sf and Sp are compared according to the rules of 4.2.4, and pseudo-codewords are copied from Cf into Cp for those fields required by the sub-schema.

(iv) Various input buffers are allocated for physical file manipulation.

(v) The file area is allocated in the DRS.

(vi) The codewords in Cp (which are now pseudo-codewords) are transformed into valid codewords by the same mechanism used for part (v). of opening write-files. This gives rise to the sort of structure illustrated in Fig. 4.12, "A Sub-Schema".

Opening core-files is similar to write-files except that steps (ii) and (iii) are omitted.

Opening update-files requires a similar set of operations to those required for read-files except that this is preceded by a copy of the file headers from the input file directly to the output file.

Chapter 5 : Summary and Conclusions

5.1 CAS System Summary

5.2 CRH System Summary

5.3 Conclusions

Section 5.1 CAS System Summary

The aim of the Content Addressable Store system described in Chapter 3 was the provision of full content-addressing facilities by using special purpose hardware to effect significant performance gains over existing software implementations. The integrated approach to system design required consideration of existing software systems and techniques for supporting content-addressing, user requirements, and existing hardware for content-addressing.

The survey of existing systems described in 2.2 and 2.3 indicated that :- simple content-addressing facilities (indexing) were provided by virtually all Data Management Systems and that full content-addressing by "selection criteria" was provided by the Information Retrieval systems and the non-procedural Data Base Management systems; the DBTG proposals allowed multiple indexing but not selection criteria; the "Relational Model" and its associated language provided exclusively content-addressing of a more complex form than pure selection criteria (but which could be decomposed into the latter); Storage-structures for content-addressing were found to be complex and inefficient on current hardware, these were discussed in 2.3 and in more detail in 3.1. Later in this section the performance of the inverted storage structure is contrasted with that of the CAS.

User-requirements were gauged by the survey of 2.5 from which a number of quantitative and qualitative requirements emerged, these were listed in 2.7.

Existing hardware for content-addressing based on rotating magnetic storage was found to be unsuitable for meeting the user requirements. One device (described in Appendix E.2) had broadly similar goals to the CAS and came nearest to meeting these requirements. However, this was considered to offer less flexibility than the CAS for the reasons listed in 3.2.1.

The design of the CAS system covered in Section 3.2 does not constrain the hardware realisation of the device, although references to a head-per-track disk indicated that this hardware was a suitable candidate for extension at the present time to achieve the desired performance goals. Current trends towards higher packing density disk storage and L.S.I. for mass storage are likely to yield hardware which could be used for the CAS system with increased performance potential. The activities involved in content-addressing by exhaustive search of magnetic storage breakdown into a number of logical steps and each of these is performed by a "unit" of the CAS Controller described in 3.2.2. This structure of the controller is a logical rather than physical organisation and although each unit is directly realisable in off-the-shelf "TTL" at the current time, other hardware implementations are not excluded provided that the Controller Commands (described in 3.2.4) can be provided. The data formats described in 3.2.3 are based on a variable intra-record storage structure derived from user requirements and recognised by the hardware, alternative data formats could be adopted if necessary without invalidating the logical structure of the CAS Controller. A software control system (Device Handler) for the CAS was outlined in 3.2.5 and the form of task specifications was described in 3.2.4.

Section 3.3 described a language (ECAL) for interacting with the CAS and which constitutes an initial user interface. The language could be used either interactively or by stored program, both modes being executed interpretively. The ECAL language was intended to demonstrate and experiment with the facilities of the CAS. The most important ECAL construct was the "SEARCH" command since this corresponded to the content-addressing search tasks of the CAS controller (the transformation of a SEARCH command into the corresponding task was illustrated in 3.2.4.5). Other ECAL facilities provided a relatively simple context within which the SEARCH command could be used. The particular example of Fig. 3.12 indicated very frequent use of "range" values in selection criteria e.g:-

```
AVPRICE   BET   1, 59999
```

Using the current definition of the ECAL SEARCH command this would have to be written:-

AVPRICE GE 1 AND AVPRICE LE 59999

Extensions of the ECAL language to permit the former construct could easily be accommodated.

The performance of the CAS was examined and predicted in Section 3.4. The CAS performance was investigated from two points of view firstly, to evaluate its advantages in full content-addressing by selection criterion as compared with an existing software implementation and secondly, to examine its performance when used for simple indexing in a transaction processing environment. In both cases there appeared to be significant advantages as outlined later.

Section 3.5 discussed the use of the CAS in conjunction with a conventional direct access store (hybrid system) and also covered some of the qualitative requirements which emerged from the user survey to ascertain whether the existence of the CAS introduced advantages or obstacles in meeting these requirements. A hybrid system is attractive because the total size of the data base could be an order of magnitude greater than the CAS (for many applications) and the characteristics of the two devices could be traded-off by the system designer.

There are a number of areas in which the CAS is expected to offer advantages over existing equipment and software techniques, these are discussed under the appropriate headings which follow together with the main disadvantages. For discussion the CAS capacity is assumed to be 10^8 bits and to be completely scanned in 1 second - figures which are realistic in terms of current technology. The data base size, if held entirely on CAS storage, would be $\simeq 10^7$ bytes allowing for unused space because variable records are used. In a hybrid system the data base could be larger, a figure of 10^8 bytes appears reasonable for many applications.

Response-Time

The CAS parallel processing ability means that response time is unaffected

by loading until the device is near saturation point when queuing effects reduce performance. For simple accesses to a single data base record (single-hit tasks) the response-time is uniformly distributed in the interval 0-1 sec. This distribution is unaffected until transaction rates of over 75% saturation cause a shift towards longer response times (as indicated in Fig. 3.14) due to time spent queuing for service. For general content-addressing operations (multi-hit tasks) the response time is 1 sec (i.e. a complete scan) and is independent of the complexity of the selection criterion up to a maximum dictated by the capacity of the K.S.U. The performance evaluation described earlier suggests that an inverted structure provides an equivalent response time for selection criteria of about 3-5 triples but its response is approximately linearly proportional to the number of triples. Furthermore, the effects of parallel evaluation of selection criteria are likely to appreciably increase response time since the direct access store was already approaching saturation for the single selection criteria examined. Response times for the multilist storage structure are generally an order of magnitude greater than for the inverted structure.

Transaction Rate

The transaction rate which can be accommodated is dependent on the complexity of the selection criteria for each transaction since the number of "key registers" in the CAS controller is fixed. The results from the queuing model of 3.4.3 indicate that a CAS containing about 32 key registers could service rates in the region of 50 simple transactions per second with negligable performance loss due to queuing effects. The highest rate of which the author is aware using conventional direct access hardware is 20 transactions per second and in this case the limiting factor is the retrieval of index material from secondary storage - a bottleneck eliminated by the CAS. Transaction rates for content-addressing of multiple data base records are more difficult to quantify. However, in view of the number of I/O transfers required for the single selection criterion in the inverted

storage structure it is probably fair to say that appreciably higher rates could be accommodated by the CAS than by using the conventional system.

The parallel execution of many search tasks will probably give rise to a higher degree of multiprocessing than that associated with current systems. This is likely to be the limiting factor on transaction rate since the hardware modularity allows additional key registers to be added. The removal of the index retrieval bottleneck is a significant factor in the achievement of high transaction rates.

I/O Channel Loading

The data channel to the CAS will be lightly loaded since it is only involved in the transfer of selected fields from selected records. This contrasts with the inverted strorage-structure which required appreciable index material transfer in addition to whole data base record transfers. The multilist storage structure is generally appreciably worse in I/O channel usage than the inverted structure.

Central Processor Loading

Since the CAS operates autonomously it is expected to require significantly less c.p.u. involvement than either the inverted or multilist storage-structure. The inverted structure is particularly problematic in its c.p.u. usage, largely due to the address list intersections which have to be performed. C.p.u. usage is confined to the operation of the CAS Device Handler. Additional loads may also be introduced because of the degree of multiprocessing mentioned earlier.

Central Memory Requirements

Central memory requirements relate primarily to buffer storage and queue storage. These overheads are comparable with the multilist storage structure and appreciably less than the inverted storage-structure which uses large amounts of core store for holding indexes and address lists.

Data Base Record Insertion

This operation is very simple in the CAS, involving the writing of a single record into CAS storage by a direct access operation. The inverted storage-structure is particularly problematic in this respect as the results shown earlier indicated (a single record insertion required 150 I/O transfers in the example chosen). The inverted storage-structure was rejected by Bell Labs (see A.6) mainly on this count.

Storage-Structure Simplification

The CAS requires a more elaborate Device Handler than current direct access stores, coupled with a relatively simple store management scheme for its variable records. Beyond this it requires no other storage-structures for the content-addressing.

The inverted storage structure is appreciably more complex. The use of the structure in content-addressing is quite complex, involving directory searching, writing address lists into temporary storage, performing intersection operations on the lists and finally retrieving the desired records. Store management and overall maintainance is also problematic because of requirements for structuring the indexes and maintaining correct sequencing of index entries for rapid search.

The multilist storage structure, although not nearly as complex as the inverted structure, still requires more complex store management than the CAS to deal with the general list structuring of secondary storage.

Main Disadvantages

The main disadvantages arise from the relatively long time (~1 sec) to perform any addressing operation on the CAS. While this is completely acceptable for the evaluation of complex selection criteria it does have certain disadvantages when simple accesses are involved:-

(a) High rates of simple access requires a high degree of multi-programming on the part of the c.p.u. with attendent overheads.

(b) If the content-addressing facilities of the CAS were used to support a network data-structure the following of an individual thread through the network would take an appreciable time as compared with implementations on a direct-access device.

Section 5.2 CRH System Summary

The goal of the CRH system described in this thesis was the provision of format-independence for use in simple batch-processing, the implementation taking advantage of an advanced machine architecture. The integrated approach to system design described in the introductory chapter required consideration of user requirements, existing software and hardware. User requirements were investigated in the survey of 2.6. Existing software for batch-processing applications (at the time of the investigation) comprised mainly the Data Processing Languages which did not provide format-independence. Hardware suitable for the implementation of format-independence existed in the form of the Basic Language Machine.

The CRH system languages (described in 4.2) and their implementation (described in 4.3) were constructed primarily as an experimental vehicle for investigating the provision of format-independence. Apart from the record description (schema) facilities and the format-independence, the other facilities and the format-independence, the other facilities were made as simple as practicable to ease implementation effort. Limitations both of language and implementation of the current CRH system which make it unsuitable as a production Data Management System are listed later. Security of the implementation was an important consideration, this is discussed after the following descriptions of the "implementation effort" involved and "overheads" introduced as a result of the dynamic linkage of program and data.

Implementation Effort

The CRH system was implemented using a software simulation of the Basic Language Machine. The simulator and the Basic Language Operation System (which included the assembler) were provided by ICL via their "Research and Advanced Development Organisation" (RADO). The simulator was suitable for "1900 Series" machines and was used at Queen Mary College, London and ICL RADO Labs in the development of the CRH system. The performance of the

simulator was estimated at 1/100 of that of the Basic Language Machine for in-core processing.

The current implementation comprises about 3,300 instructions which required an estimated 250 hours of 1900 machine-time to write and debug. The use of a simulator was inconvenient from two points of view. Firstly, its heavy demands on c.p.u. time and long execution-time meant difficulties in fitting in with the normal job streams at Q.M.C. with the result that remote (overnight) runs were the norm. Secondly, there were reliability problems since the Basic Language Operating System was not fully debugged and complications were experienced in the interaction of the simulator with the George II Operating System under which it ran. As a result computing facilities were at a premium and meant that the code produced was non-optimised. An attempt at re-coding part of the CRH system indicated that a reduction in the region of 30% might be possible by complete re-coding. The system was debugged to a state considered practical for experimental purposes, the examples of 4.2.5 were executed satisfactorily.

Overheads

There are overheads in terms of store occupancy and c.p.u. usage. Store occupancy overheads occur mainly in the presence of the Coded Data Description at run-time. These overheads could be removed as mentioned later. C.p.u. overheads arise because of the dynamic linkage of program and data when a file is opened. These overheads would be reduced by the code optimisation mentioned earlier. A further, more significant reduction (as adopted for NIMMS and IMS) would be to perform the linkage operation selectively rather than each time a file is opened, this point is discussed later. Although a special Basic Language Operating System variant existed for performance monitoring it was not possible to use this to monitor c.p.u. overheads because of software incompatibilities. An estimate was made of c.p.u. overheads for the following situation:- the file had three record schemas each with fifty items and the program required all items in all record schemas.

In this case the overhead was about 100,000 instructions in opening the file (less than one second c.p.u. time in the Basic Language Machine). The most significant contribution to this figure arose from the searching of the Coded Data Descriptions which was linearly proportional to the number of items declared in the sub-schema (e.g. if half the items were declared the overhead would be 50,-60,000 instructions).

Security

An important aim implementation of the CRH system was to provide security of facilities against abuse or erroneous usage wherever possible. The features which contribute to the security of the implementation are:-

(i) Seperation between names declared in the host language and the CRH objects of the data declaration.

(ii) Compile-time checking of parameters to CRH functions (ensuring parameters are correct for context). Compile-time checking of complete item names.

(iii) Run-time error detection if DML is used on the objects of a file schema before the file is opened.

(iv) Hardware bound-checking (inherent in codewords) on those fields known to a user program.

(v) Dynamic linkage of program and data increases integrity by minimising the risk of wrongly interpreting a file contents.

Certain points however, detract from the overall security, these are:-

(a) overlapping record areas

(b) update-files require the user to output (write) unrecognised record instances instead of performing this automatically.

Point (a) above opens the system to abuse since, although the return of control following execution of the READ function (see Chapter 4, Fig. 4.3) indicate to which record class the current record belongs a user may still use the items of a different record schema to access this record. Two alternative solutions are possible - either a separate record area is allocated for each record class, or the codewords corresponding to other record schemas are made undefined following the identification of the class of the current record. These solutions result in either increased storage or c.p.u. overheads respectively. Point (b) above means that the security of the update operation is dependent on the user. This situation could be rectified either by minor modifications to the READ function so that unrecognised records are automatically output or more simply by providing a semaphore in the file area to ensure alternating use of the READ and WRITE functions. If the latter course of action were adopted it would have to be combined with one of the solutions to point (a) to ensure that unrecognised records could not be corrupted when in the file area.

Limitations and Improvements

There are a number of improvements which could be made to the CRH system to overcome limitations of the current implementation. These improvements were not effected because the CRH system was intended as an experimental system rather than a production Data Management system. The improvements listed below separate into two classes, (i) through (v) are improvements to the implementation of the CRH system, (vi) and (vii) are extensions to the CRH system.

(i) Elimination of the implementation features i.e. the "TYPE" declaration (see E.19), the item "CON" for control bytes (see E.20) and the INIT function (see E.18) currently required for variable records.

(ii) Store-recovery facilities to recover file areas from files which have been closed. Optional overwriting of the Coded Data Description by the corresponding File Area to reduce store occupancy.

(iii) Ability to input "synonyms" for names in the program Data Description prior to opening files at run-time.

(iv) Communication of status conditions to user program from DML functions instead of taking the standard Basic Language error actions (see E.22) on run-time errors.

(v) Separation of program object code and Data Reference Structure into different segments so that the former becomes re-entrant.

(vi) Extensions to the DML functions to allow:-

(a) Multi-reel files

(b) The use of hardware features of the magnetic tape decks e.g. backspace, skip to tape-mark.

(vii) Ability to "dump" a linked Data Reference Structure so that the linkage need not be performed each time a file is opened.

Point (viii) above is significant because it provides the facility to comprise between the full run-time linkage initially envisaged for the CRH system and the compile-time linkage provided by the Data Processing Languages (a dump of a linked CRH user program being equivalent to an object program produced from the Data Processing Languages). If such a facility were provided it would have to be supported by a software management system for controlling dumping and re-use of programs. Run-time checks would have to be performed to ensure that file format corresponded to that assumed by the dumped program in order to preserve the degree of integrity which is intrinsic in the current dynamic linkage of the CRH system.

Section 5.3 Conclusions

The two systems described in this thesis were developed using the integrated approach to system design referred to in the introduction. Consideration of user requirements, existing software and hardware provided a suitable basis for establishing the relevance and potential advantages of the research systems.

It is believed that the CAS system proposed for content-addressing overcomes many of the disadvantages of content-addressing associated with current software implementations, providing advantages in those areas listed in 5.1. The high degree of parallelism in the CAS hardware makes it suitable for application to areas which do not require the full content-addressing capabilities but which have a high frequency of simple accesses to a data base (transaction processing). The performance potential of the CAS in this environment appears considerably in advance of existing systems. It is highly likely that further applications of the CAS (or derivatives) will be found to other problem areas outside the scope of this thesis.

Probably the two most important current proposals for future Data Base Management software are those of Codd's Relational Model, and the CODASYL DBTG proposals. The content-addressing facilities of the CAS would be of particular benefit to implementations of the DSL Alpha used with the Relational Model. The qualifier expressions permitted in DSL Alpha giving rise to multiple CAS search tasks. The CAS could also be advantageously applied to implementations of the DBTG proposals, directly supporting the multiple indexing facilities provided by the latter.

The conclusion drawn from the investigations is that the CAS system is widely applicable, appearing to fulfil user requirements for content-addressing without burdening the Data Management software with complex storage-structures and provides significant performance benefits over current software implementations.

The CRH system investigations show that it is possible to provide format-independence within the context of a relatively simple Data Management system, taking advantage of the advanced architecture of the Basic Language machine for speed and security of implementation. The conclusion drawn is that by carefully selecting a particular band of applications and machine architecture it is possible to selectively apply important Data Base Management features in areas where a complete data base implementation would not be appropriate.

In terms of future development the two systems (CAS & CRH) differ considerably. It is unlikely that the CRH system or derivatives will be used commercially since the Basic Language Machine project has been discontinued. The CAS system is more significant and widely applicable and the work of this thesis is being continued and extended with the aim of constructing prototype CAS hardware under a normal cost sharing contract between ICL and ACTP. It is felt that the introduction of classes of device like the CAS is likely to have a beneficial effect on the longer term development of Data Management Systems provided that all important parameters like cost and reliability can be made as attractive as the logical capabilities and performance indicated in this thesis.

Bibliography

1. "Direct Access Techniques" ICL Technical Publications Ref. 4107.

2. "IBM Data Management Services" Document GC26-3746 IBM Systems Reference Library, 1971.

3. "Operating Systems George III and IV" ICL Technical Publications Ref. 4169, 3 ed., Aug. 69.

4. Cox N.S.M, Gross M.W, Editors "Organisation and Handling of Bibliogrphic Records by Computer" Proc. Seminar at Newcastle University, ORIEL Press, Newcastle 1967.

5. Daley R.C, Neumann P.G. "A General Purpose File System for Secondary Storage" Proc. AFIPS, 1965 Fall Joint Computer Conference, Vol. 27, Nov. 1965.

6. "Introduction to ICL COBOL" ICL Technical Publications Ref. 4118, 2 ed., May 69.

7. Hendry D.F, Mohan B (1968). "A BCL Manual", Internal Report No. ICSI 103, University of London Institute of Computer Science.

8. "IBM System 360 Operating System, PL/1 Language Reference Manual" Form C28-8201-2, 3 ed., Oct. 1969.

9. "The Nebula Reference Manual", ICL Technical Publications.

10. Backney A.B, Scratcherd M. "A Survey of Information Projects", March 1969, Information Collection and Retrieval Sub-Committee, National Engineering Laboratory, East Kilbride, Glasgow.

11. Blackney A.B, Scratcherd M. "Information Retrieval in the U.S.A.", January 1970, Information Collection and Retrieval Sub-Committee, National Engineering Laboratory East Kilbridge, Glasgow.

12. Fraser A.G. "Integrity of a Mass Storage Filing System" Computer Journal Vol. 12 No. 1 Feb. 1969.

13. Nance J.W, Lathrop J.W. "System Design Specifications, General Purpose ORBIT", Ref. TM-DA-20/000/00, System Development Corporation, 2500 Colorads Ave, Santa Monica, Calif.

14. "PEARL, Preliminary Specifications" ICL Technical Publications Ref. K-ADN37.

15. CODASYL "A Survey of Generalised Data Base Management Systems', May 1969, ACM H.Q.

16. CODASYL "Data Base Task Group Report, October 1969" and April 1971, ACM HQ

17. C. C. Halatsis "Hardware Considerations of Executive and Information Retrieval Systems" Ph.D Thesis, Manchester University. August 1971.

18. "PLUTo System" ICL Technical Publications Ref. 4205, 2 ed., April 1970.

19. "Integrated Data Store Reference Manual" File Number CPB-1565B, Honeywell Information Systems Inc., 1971.

20. "Information Management System IMS/360", Application Description Manual H20-0524-1, IBM U.K.

21. "NIMMS File Handler" ICL Technical Publications Ref. 4243, Jan. 1971.

22. "TS/DMS User's Guide, Commercial Data Management System", Ref. TM-4132/000/02A, System Development Corporation, 2500 Colorado Ave, Santa Monica, Calif.

23. "Generalised Information System GIS/360", Application Description Manual H20-0521-1, IBM U.K.

24. Olle T.W "A Non-Procedural Language for Retrieving Information from Data Bases" IFIP Congress, Edinburgh, August 1968.

25. Bleir R.E "Treating Hierarchical Data Structures in the SDC Time-Shared Data Management System", Proc ACM 22nd National Conference, MDI Publications, Wayne, Pa (1967).

26. Bleir R.E, Vorhaus A.H. "File Organisation in the SDC Time-Shared Data Management System" Ref. SP-2907, System Development Corporation, 2500 Colorado Ave, Santa Monica, Calif.

27. Lefkovitz D. "File Structures for On-Line Applications", Spartan Macmillan and Co. Ltd.

28. Hsiao D, Harary F. "A Formal System for Information Retrieval from Files" CACM, Vol. 13, No. 2, Feb. 1970.

29. Codd E.F "A Relational Model of Data for Large Shared Data Banks" CACM Vol. 13, No. 6, June 1970.

30. Codd E.F. "A Data Base Sublanguage Founded on the Relational Calculus" Proc. 1971 ACM SIGFIDET Workshop.

31. Feldman J.A, Ravner P.D. "An Algol-Based Associative Language" CACM Vol. 12, August 1969.

32. Gray J.C. "Compound Data Structures for Computer Aided Design: A Survey", Proceedings ACM 20th Anniversary Conference, 1967.

33. "The FIND II Multiple Enquiry System" ICL Technical Publications Ref. 4187.

34. Hanlan A.E. " Content-Addressable and Associative Memory Systems, A Survey" IEEE Trans. EC-15, 4 (Aug. 66).

35. Lum V.Y "Multi-Attribute Retrieval with Combined Indexes, CACM, Vol. 13, No. 11, November 1970.

36. IBM System 360 Component Descriptions "2314 Direct Access Storage Facility and 2844 Auxiliary Storage Controller" Form A29-3599-3, 6th Ed., 1970.

37. U.K. Patent 1.1718.429 "Data Processing System" August 1967.

38. Parker J.L. "A Logic-per-Track Retrieval System", IFIP Congress. August 1971.

39. Coulouris G.F., Evans J.M., Mitchell R.W. "A Hardware-Aided Approach to Content-Addressing in Data Bases", Hardware Software Firmware Trade-offs, Proc. 1971 IEEE Conference.

40. Coulouris G.F., Evans J.M., Mitchell R.W. "Towards Content-Addressing in Data Bases" Computer Journal Vol. 15, No. 2, May 72.

41. Cox D.R, Smith W.L. "Queues", Methuen, 1961.

42. Iliffe J.K. "Basic Machine Principles", Macdonald 1968.

43. Iliffe J.K. "Elements of BLM" Computer Journal Vol. 12, No. 3 August 1969.

44. Hawk A.E, Dent B.A. "Burroughs B6500/B75000 Stack Mechanism" AFIPS, Proc. SJCC, 1968.

Appendix A

Real-Time User Reports 1970

A.1. System Development Corporation

A.2. University of Illinois

A.3. Stanford University, Department of Computer Science

A.4. California Department of Motor Vehicles

A.5. Boeing Scientific Research Laboratories

A.6. Bell Telephone Laboratories

A.7. National Aeronautics and Space Administration

A.8 University of Chicago

A.1 SYSTEM DEVELOPMENT CORPORATION

Address: 2500 Colorado Avenue,
Santa Monica,
California.

Activities

The System Development Corporation undertakes a diversity of software research and development. Our prime interest was in a time sharing system called TS/DMS which includes a system called CDMS for flexible data management. The next part describes these in more detail, and a demonstration was provided of the use of CDMS (see later).

Systems Discussed

The TS/DMS system is offered commercially on a time-sharing basis and comprises

TS/EXEC	Time Shared Executive
/JOVIAL	JOVIAL Compiler
/TINT	Interactive Language Interpreter
/ORBIT	On-line retrieval of bibliographic information
/PROPAC	Programmers Package (Variouslanguages)
/CDMS	Commercial Data Management System, previously called TDMS.

The system is capable of supporting up to twenty simultaneous users on a 360/67 at The SDC Datacentre in Santa Monica. Our prime interest was in the CDMS system which provides a real time data management capability. This system has however been withdrawn in order to develop it further and will later be reissued. It had been found that users were not committing themselves fully to the system and integrating it into their businesses. On reissue a greater degree of system support will be provided by SDC. The system development will include modifications enabling it to run under OS 360 which should greatly increase its saleability. Literature was provided on CDMS and two other systems, DSI and ORBIT. The DSI system is separate from TS/DMS and has fixed length records, it is designed for less complex problems than CDMS. We discussed the CDMS system in detail.

The CDMS system provides high level facilities for storing and manipulating data where records appear to the user to be content-addressable. No special purpose hardware is used. The system has a

flexible hierarachical record structure comprising data fields which may be variable or fixed and vectors (repeating groups) containing a variable number of fields. A data base in the system is a collection of similarly structured records, users normally have a number of different data bases. Each data base if fully inverted so that records are content-addressable on any field or combination of fields.

Fields can be compared with key values during a search in a variety of ways equal, greater than, less than or equal, etc. CDMS had a number of "operations" which are program segments each allowing the user to perform a particular set of functions.

DEFINE	used for defining the record structure of a data base
GENERATE	used to put data into the data base
QUERY	used to retrieve, manipulate and print data from a particular data base.
COMPOSE	a report generator for formatted output
UPDATE	used to dynamically update the data base
SCOUT	similar to QUERY but with extra facilities for use with the IBM 2260 text display.
MAINTAIN	used for data base maintenance, copy, cleanup.

There are two ways of updating a data base. The real-time version is via the UPDATE operation and a batch update is available under MAINTAIN. There is no lockout at record level. All operations involving update require total access to a file which prevents other users reading or accessing the file in any way. The real time update involves overwriting existing information and the system does not store the previous record before carrying out the update. The system designers were of the opinion that users did not use real-time updating because the overwriting of existing information gave audit problems. In the case of temporary data bases the real time update could sometimes be used effectively. e.g., on-line error correction of a data base used to accumulate transactions for a subsequent batch update. File security against system failures must be provided by the user, the utility EMOV allows disk files to be copied to magnetic tape and can be used to provide a backup for recovery.

Demonstration of CDMS

Two aspects of the CDMS system were demonstrated, the infromation retrieval ability known as the "Query" operation and the report generator

known as the "Compose" operation, both on a single data-base called TRAFFIC 2. A teletype terminal connected to the SDC 360/67 at Santa Monica was used; there were eleven other users logged into TS/DMS. A listing of the demonstration is included below, certain lines have been preceded by an integer for ease of reference in the text which follows. A record description for the TRAFFIC 2 data base was requested and from this various search requests were formulated. A magnetic tape file falled "Crash" contained a "COMPOSE" definition of a report; this file was loaded on to disk and used to produce a report from the TRAFFIC 2 data base. The following notes refer to the lines with corresponding numbers appearing on the left-hand side of the listings.

Line

1 user request for the Query operation
2 the data base name is input
3 used to find how many other TS/DMS users are operative
4 user defined limit on print out
5 user requests a record description
6 the system having output 5 lines asks if it may continue
7 requests a description of the range of values of the variable C2
8 user request to print every third value of the variables V1...V14
9 system outputs the frequency of occurrence, the "V" number, the actual value
10 cancels the search loop
11 print the type of collision where C89 has the value alcohol, the lines which follow up to the line marked 12 illustrate error detection and various retrieval requests.
12 user requests EMOV in order to load a report generating file CRASH from magnetic tape on to the disk.
13 delete EMOV, the CRASH file has been loaded
14 requests COMPOSE
15 the data base TRAFFIC 2 is entered.
16 the CRASH file is named as the Source of the report Specification
17 this command is actually RUN, the "S" has been produced by a line error. The report generation follows
18 lists the report specification contained on the CRASH file
19 this provides a "proof" of the report format

```
    TS/DMS VER 7.6 MOD 19B CH 72
    /LOGIN CORP 1130015
    LOGGED IN  11:16  3/18/70
1   /QUERY
    PGM.STARTED 3A000
    'QUERY' VERSION 6.0 IS OPERATING.
2   ENTER DATA BASE FILE IDENTIFIER: TRAFFIC2
    NEXT
    U
    ** 'U' IS NOT A QUERY COMMAND.
3   /USERS
    11 USERS
4   LINES 5
    NEXT
    COLUMN
    NEXT
5   DESCRIBE
    ----

        C1  DOCUMENT RETRIEVAL NO. (NUMBER)

        C2  TYPE OF COLLISION (NAME)

        C3  WEATHER (NAME)

        C4  LIGHTING (NAME)

        C5  ROAD CONDITIONS (NAME)
6   CONTINUE? (Y/N): Y


        C6  TIME OF ACCIDENTS (REPEATING GROUP)

       C61      DATE (DATE IN 6)
       C62      MONTH (NAME IN 6) FORMAT IS XXX
       C63      HOUR (NAME IN 6) FORMAT IS XXXX
       C64      DAY OF WEEK (NAME IN 6) FORMAT IS XXX
    CONTINUE? (Y/N): N

    ----
7   SHOW C2
    ----

         3   V1  ANGLE COLLISION
         8  V14  SIDESWIPE


8   SEARCH: V1...V14/3

9        3   V1  ANGLE COLLISION
         4   V4  BROADSIDE SWIPE
        24   V7  HIT OBJECT
         9  V10  OVERTAKING SIDESWIPE
        17  V13  REAR END
    CONTINUE? (Y/N):N

10  SEARCH: S

    ----
    SHOW C89
    ----

        13  V1  ALCOHOL
        18  V2  DISREGARDED SIGNAL
         9  V3  FOLLOWING CLOSE
        94  V4  NOT STATED
        22  V5  SPEEDING
```

```
----
PRINT C2 WHERE C89 EQ V1
----

                C2

BROADSIDE

OVERTURNED

IIT OBJECT

NON-COLLISION

REAR END

CONTINUE? (Y/N): N
----
PRINT SUM C1 WHERE C89 EQ V1
----

SUM DOCUMENT RETRIEVAL NO. = 1921.

----
SUM C89
** 'SUM' IS NOT A QUERY COMMAND.
PRINT SUM C89
**ERROR AT 'C89'.

11. PRINT C2,C61,C63,C83 WHERE C89 EQ ALCHOL

'21' SPACES IN EXCESS,THE REFORMAT OPTIONS ARE: 'ANGLE', 'UNBLOCK', N
  EXCESS', OR '$'-
REFORMAT: $
NEXT
PRINTC2,C62
** 'PRINTC2' IS NOT A QUERY COMMAND.
PRINT C1,C61,C63,C83 WHERE C89 EQ AC
----

NO DATA QUALIFY

----
PRINT C1,C61,C63,C83 WHERE C89 EQ ALCOHOL
----

C1      -C6-
        C61 . . C63

103  05/02/69  1015

                        . . . . . -C8-
                                  C83

                        NONE

108  05/12/69  1730

                        SCRATCHES

110  05/11/69  130

CONTINUE? (Y/N): N
```

```
    ----
12  /EMOV
    PGM.STARTED 44000
    EMOV, VERSION EM08 IS OPERATING
    INPUT DEVICE-DISC/TAPE
    TAPE
    TAPE FILE ID-NAME SN VTNNNN DC
    CRASH S2 V96270
    TAPE VOL 6270 TO BE MNTED ON.C7




    FOLLOWING FILES ARE ON TAPE
      NAME       FORM
    PROPERTY        1
    CRASH           1
    PRICE1          1
    ENTER 'RUN' INSTEAD OF FILE ID TO TERMINATE LIST OF FILES
    DISC FILE ID-NAME SN FN VTNNNN N/E
    CRASH S2
    CATALOGER ERROR-FILE NOT FOUND
    DISC FILE ID-NAME SN FN VTNNNN N/E
    CRASH S2 F1



                N
    FILE CRASH    , 2314 VOL. 9003 OPENED ON A2
    DISC FILE ID-NAME SN FN VTNNNN N/E
    RUN
    PGM.STARTED...STANDBY
    CRASH     COPIED
    PGM.CONCLUDED
    INPUT DEVICE-DISC/TAPE
13  /QUIT EMOV
    PGM.QUIT
14  /COMPOSE
    PGM.STARTED 3CC00
    'COMPOSE' VERSION 6.0 IS OPERATING.
15  ENTER DATA BASE FILE IDENTIFIER: TRAFFIC2
16  ENTER COMPOSE FILE IDENTIFIER OR 'NONE': CRASH
    NEXT
```

```
17  SUN
    REPORT GENERATION HAS BEGUN.   03/18/70  11:41
    DATA BASE NAME IS...   SAMPLE DATA BASE TRAFFIC STUDY

                TRAFFIC REPORT FOR FIRST WEEK OF MAY, 1969
                     OAKLAND     PAGE NUMBER:      1
                   INFORMATION BY TYPE OF VIOLATION


    VIOLATION              DATE OF      DRIVER NAME           ROLE
                           ACCIDENT

    ALCOHOL                05/02/69     WARREN MASSEY         DRIVER

    TOTAL ACCIDENTS        1

    DISREGARDED SIGNAL     05/03/69     ARTHUR PLATT          DRIVER
    DISREGARDED SIGNAL     05/04/69     BEVERLY BRANDON       DRIVER

    TOTAL ACCIDENTS        2

    SPEEDING               05/01/69     MARY G LUTHER         DRIVER
    SPEEDING               05/02/69     ROBERT DE PAUL        DRIVER
    SPEEDING               05/03/69     MONA R LISSA          DRIVER
    SPEEDING               05/04/69     JUDITH L MORAN        DRIVER

    TOTAL ACCIDENTS        4

    TOTAL NUMBER OF ACCIDENTS      7


    REPORT GENERATION IS FINISHED. 03/18/70  11:44

    PROGRAM IS STOPPING.  TYPE "/GO COMPOSE" TO CONTINUE.
   PGM.STOPPED
   /GO COMPOSE
   PGM.STARTED
   'COMPOSE' VERSION 6.0 IS OPERATING.
   ENTER DATA BASE FILE IDENTIFIER: TRAFFIC2
   ENTER COMPOSE FILE IDENTIFIER OR 'NONE': CRASH
   NEXT
18 REVIEW
        DATA BASE NAME IS SAMPLE DATA BASE TRAFFIC STUDY
        QUALIFY DRIVER VIOLATION  WHERE C89 EXISTS AND C89 NO V4 AND
        LS 5/5/69
        SORT BY DRIVER VIOLATION, DATE
   T1   T. <TRAFFIC REPORT FOR FIRST WEEK OF MAY, 1969>
   T2   T. OAKLAND, <PAGE NUMBER:>, 'P
   T3   T. <INFORMATION BY TYPE OF VIOLATION>
        FEED 4 AFTER
   H3   H. VIOLATION, DATE OF* ACCIDENT, DRIVER NAME, ROLE
        FEED 4 AFTER
   C2   C. DRIVER VIOLATION, DATE, NAME, ROLE
        MASK C2F1 = XXXXXXXXXXXXXXXXXX
        MASK C2F3 = XXXXXXXXXXXXXXXXX
        MASK C2F4 = XXXXXXXXXX
   R1   R. OVERALL =  'L TOTAL NUMBER OF ACCIDENTS, COUNT DOCUMENT
        RETRIEVAL NO.
        MASK R1F2 = 009
   R2   R. DRIVER VIOLATION = 'L TOTAL ACCIDENTS, COUNT DOCUMENT
        RETRIEVAL NO.
        MASK R2F2 = 009
        LINK H3, C2
        FOR EOF T1, T2, T3, H3
   NEXT
```

```
19  PROOF

        ....5....0....5....0....5....0....5....0....5....0....5....0....5....0....5

    T1                TRAFFIC REPORT FOR FIRST WEEK OF MAY, 1969
    T2                      OAKLAND     PAGE NUMBER:    0009
    T3                   INFORMATION BY TYPE OF VIOLATION

    I3  VIOLATION                DATE OF      DRIVER NAME           ROLE
                                 ACCIDENT

    C2  XXXXXXXXXXXXXXXXXX       99/99/99     XXXXXXXXXXXXXXXXX     XXXXXXXXXX

    R1  TOTAL NUMBER OF ACCIDENTS     009
    R2  TOTAL ACCIDENTS     009

        ....5....0....5....0....5....0....5....0....5....0....5....0....5....0....5

    NEXT
```

A.2 UNIVERSITY OF ILLINOIS

Address: Urbana, Illinois.

Activities

The Illiac 4 is a high speed array processor controlled and serviced by a Burroughs B6500. Its structure is outlined below and described in Ref. 1,2. Apart from its relevance to computer system design, its special interest to us derived from the existence of a large, fast disk file, to be used as a backing store to Illiac.

The main system of interest to us was the design of a logic-per-track disk system which related to the Illiac only by the suitability of Illiac for simulating such a system.

Logic-per-Track Disk Project (J. L. Parker)

Mr. Parker is engaged on a Ph.D. project which involves the design of a content-addressable data store. The device is envisaged as a logic-per-track disk of about a thousand tracks and a central control unit. Each disk head has an associated logic chip of less than two thousand gates which allows preprocessing of information before it is passed to the control unit. The tracks, each of 1M bits, would be searched in parallel, the whole disk being searched for a given key in a single revolution by a method described in detail below.

A simulation of the system on the Illiac 4 will be used to evaluate the design, each of the Illiac PE's will be used to simulate the head logic on a single track, the simulated system having 64 tracks. A thesis proposal outlining the project and a preliminary design was provided by Mr. Parker and the proposed system is outlined below. Data records in the system are variable - length bit strings up to a maximum of 64K bits. The large variable record format is deemed necessary for data base applications; examples quoted included Land Usage records, hospital records, insurance files and census information.

Two alternative formats are proposed for data and key field organisation; either (i) the access key is separated from the data or (ii) any fields within the data record can be used as a key. The first method (i) has the disadvantage that it cannot respond to unanticipated enquiries without reorganising the file,

but Mr. Parker was of the opinion that this method was adequate for a large class of applications. Method (ii) involves using a set of fixed length field pointers at the head of each record. The hardware must locate the record fields using the pointers before carrying out the association. This requires more complex search hardware.

If the keys are separated from the data as in (i) above, the disk will contain three record types: holes, data and access-key records. Each record consists of a two-bit type identifier, a 16-bit length field and finally the variable length bit string. The "holes" contain free space and are overwritten by data as required. Store fragmentation requires periodic garbage collection. Each data record is preceded by one or more access key records. During searching a match on any one of the access keys results in record retrieval.

Search operations are initiated by passing a search-key value and the key length to the device control unit. The disk is searched by comparing the given search-key with access-key records of the same length or larger for a match, the mechanism for this comparison and for record retrieval is described later. The device can search on all tracks simultaneously for one key at a time, two modes of searching could be employed (i) where only a single record is required and the scan can terminate after a key match, (ii) where all the records matching a given key are required, the disk must be completely scanned once. The disk surface is split into quadrants, searching may begin at any quadrant boundary thus reducing the "latency".

The logic is serial by bit, a single clock track synchronising the heads and device control unit. The head logic can perform five operations: read, write, search, search for a hole, and garbage collect. Each head logic chip contains two 21 bit registers for holding the bit address relative to a segment boundary and a bit address of the current record, and also two 16 bit registers for holding and decrementing the length of the current bit string. There is a set of logic for each of five operations which the head can perform. We will describe only the search and read operations in detail.

In searching for a particular access-key the length of the search-key is moved into the length register and the operation begins under control of the timing track. Initially searching will always start on a segment boundary but before the track is completely scanned the head may be suspended, in this case it stores the current address and resumes at this point when requested.

Access-keys which are shorter than the search-key length are ignored by skipping over them using the length value stored in a register and used to count the bits. The device control unit broadcasts the search-key bit-by-bit repeatedly to all heads beginning again on each cycle at the first bit of the search key. Since only an equal compare is required it does not matter which order the bits are compared in. This fact can be used to allow for access keys recorded at any position in a disk segment. The access-key is written using the same cyclic broadcast as for searching, and commencing to write as soon as a hole is located. If the search-key has the same length as the access-key then their phases will correspond.

With some modification, algebraic comparison would also be possible. A successful key match results in a signal to the control from the appropriate ship then all chips halt storing the address of the current record. The control unit knows which head has the address of a matching key record and can then be used to read the following data record. The read operation is performed by a single head at a time, it counts down to the location at which the head was suspended and transmits the next data record to the control unit skipping over any key records encountered. The retrieval of a record thus involves a key search which locates the record and a read operation which transfers the record from the disk.

The hardware will be supported by software organised on a functional basis providing facilities for:-

1. file: creation, deletion, restructuring.
2. record: insertion, deletion, modification.
3. retrieval of a single record which matches a given key.
   retrieval of all records which match a given key.

A.3 STANFORD UNIVERSITY, DEPARTMENT OF COMPUTER SCIENCE

Address: Stanford, California.

Activities

Professor Feldman has designed and implemented the languages AL and LEAP (CACM, August 1969) in which data is addressed associatively using hash-coding techniques. In AL, core store addresses are used in the hash-coding, LEAP has hash-coded backing store addresses where high order address bits give the page number, and low order bits give the position within the page. We discussed associative addressing generally with Professor Feldman and he gave numerous references to other groups with similar interests.

The SPIRES system for storage and retrieval of bibliographic information was discussed in detail, a demonstration (see later) was also provided.

Discussion with Professor Feldman

Associative search mechanisms can provide efficient information retrieval for a particular class of requests, there are more complex inquiries where a parallel search can only partially solve the problem. In these cases, the associative search must be supplemented by serial processing and it is this latter factor which is likely to degrade the system performance. Two examples of search requests which could not be directly expressed as a single associative search were given:

(i) Disjunctive keys: "All the people whose eyes are either blue or grey". the hardware would have to provide alternative key values.

(ii) Cross-coupled keys: "All the people whose name is the same as the head of their department". This request requires multiple searches.

Professor Feldman was familiar with the TDMS system and commented that it was a bad idea to totally invert files; an adaptive system capable of selective inversion would be better.

The SPIRES System

The SPIRES system is used for document retrieval where each record in the system contains information about a particular document. The system has high level language facilities for retrieval and records are retrieved by specifying the contents of certain data elements (fields) using a selection criterion.

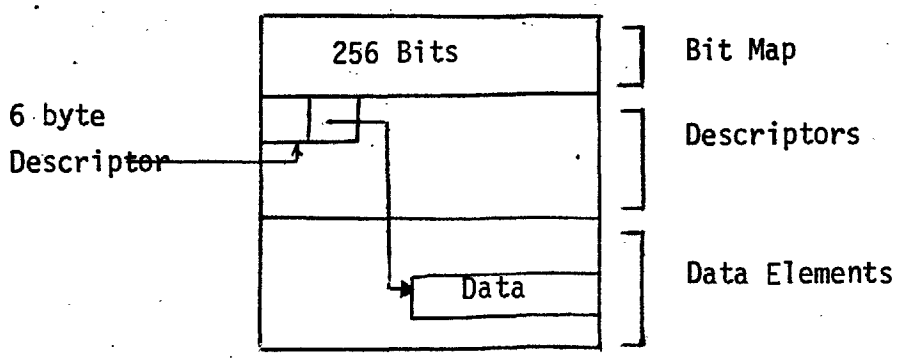
The prototype SPIRES System is available on on-line to users in several departments at Stanford for access to six small and medium-sized data bases.

The prototype system runs on the Stanford Computer Centre 360/67 as a sub-system of OS/360 occupying a single core partition and uses software generally available to Stanford users. The SPIRES system requires fast file recovery procedures but the 360/67 being a service machine oriented to a high through-put of small jobs has minimal recovery procedures following a crash (an event which occurs on average every 36 hours). Plans are being made to implement a slightly modified version of SPIRES on a separate 360/50 with 128 K bytes of core, 1 M bytes mass core store and running under OS/360. We discussed the prototype system in detail.

Each data base is stored in a 23/14 multi-spindle disk together with a set of indexes to the data records by authors, subjects, etc.

Logical records in the SPIRES system consist of 256 optionally occurring variable length data elements, each of which belongs to a specific class, e.g. Author. Records may be retrieved by specifying the values of particular data elements i.e. addressing by content.

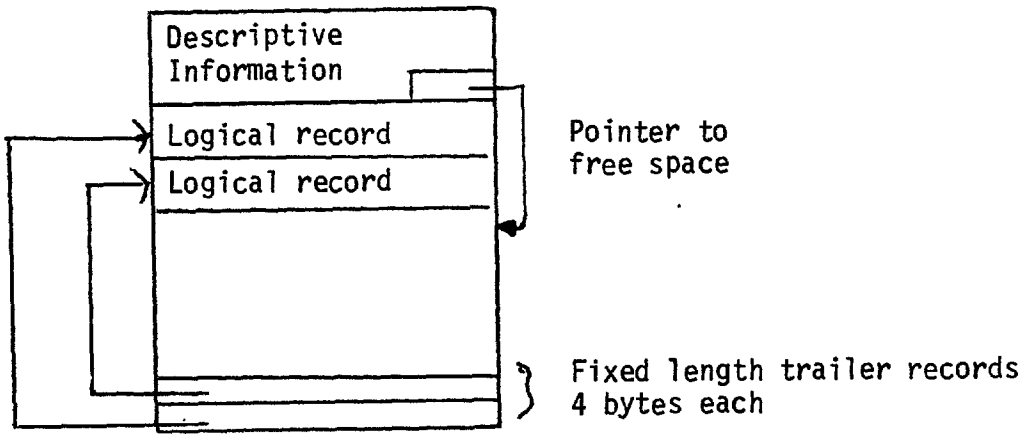
Logical Record



The bit map is used to indicate the presence of each data element. A descriptor containing position, length and data type exists for each of the 256 bits. The user has the opportunity to restrict data elements by specifying a maximum size and particular data type.

The information store consists of a data base containing logical records and a set of directories, each corresponding to a particular data element class. The data base is comprised of blocks of direct access storage, each occupying half a track (i.e. 3250 bytes).

Data Base Block



The fixed length fields provide ordered access to the individual variable length records. The store in each block is managed separately, record deletion resulting in copy operations as necessary to maintain a contiguous vector of free space. Records are between 300 and 600 bytes long and are uniformly distributed among the data base blocks.

The directories consist of one or more index blocks each having a similar format to a data base block, access to the directories is hash coded for speed. A search for Author value "Smith" would pass the value "Smith" to a hash coding algorithm which would generate the address of the "primary entry" for Smith within the Author index block. The primary entry for Smith is the head of a record address chain which may go across index block boundaries and addresses each record with Author value "Smith", this corresponds to a "multilist" storage-structure. Addresses are two part, block number and record number within block.

The system planned for the model 50 differs slightly from the system described above. Records will have certain fixed length elements in addition to the variable elements and a tree-like index structure will be used instead of simple chaining. Data protection under real-time updates is not provided on the present system but will be necessary on the next. Lockout is desirable at the data element level but the implications of this have not yet been investigated, lockout at record level is certainly required. The standard OS/360 facilities of ENQ and DEQ for protection are considered too restrictive and inefficient for this particular application. A problem with the design of the new system is the necessity for comprehensive fast file recovery procedures which impinges on the generalised file structure. The kind of recovery procedures proposed is some kind of transaction catching facility below the level of OS/360.

SPIRES Demonstration

A print-out of the demonstration carried out on an IBM 2741 typewriter terminal may be found on the pages which follow ; certain lines have been marked by preceding them with an integer in order to refer to them in the text which follows. To distinguish user messages from system messages, the former are printed in lower case characters and the latter in upper case followed by a question mark if a user response is expected. Record print-outs may use upper or lower case characters, and format may be either user defined or a standard format. In the demonstration, an attempt to print a record using the standard format failed. Records are located by specifying the values of certain data elements. Conceptually, the search is multilevel, at each level it is applied to the set of records located by the previous search, at the top level all records in the data base are searched. Facilities are provided to return to the previous level (backup) and to backup to the top level (restart). The single level backup did not work in the demonstration.

Line

1. The system requests search keys for a search at the top level.
2. The user message says that a format specification follows on subsequent lines.
3. "type own" prints the records retrieved by preceding operations, in the user defined format.

4, 5 The "backup" command had no effect and a "restart" was issued to return to the top level.

6. A search at the top level is initiated.

7. A further search of the records retrieve by 6.

8,9 The "backup" command has no effect.

10. An attempt to initiate a further level of search fails.

11. "restart" is issued which returns the system to the top level.

12. Erroneous message input, the following lines give diagnostics and after the SEARCH REJECT message the system is ready to search at the top level.

13. The '##' character indicates termination, any word with MESON as the first five characters would match.

14. The "type" command failed to output the record in the standard format, instead output was in the user defined format.

15. This and the following lines give a syntax error and a search for records with a data field greater than a given value.

```
** Welcome to SPIRES
For additional information, type SHOW NEWS.
SEARCH ? show news

     New information added to the NEWS file as of 11/21/69:

       1.  The output facility has been extended to include:

                    element selection
                    format selection
                    text lin...
SEARCH ? preprint
1  FIND? title muon and title pion
   TITLE SEARCH FOR... MUON
   TITLE SEARCH FOR...  PION
           1  DOCUMENT(S) ACCUMULATED.
2  ? choose elements
   ELEMENTS ? author
   ELEMENTS ? title
   ELEMENTS ?
   TO USE THIS FORMAT, ENTER: TYPE OWN.
3  ? type own

   TITLE                    ANALOGS OF GIANT RESONANCES FROM PHOTO-PION PRODUCTION AND FROM MUON AND PION
   AUTHOR                   Uberall, H.

4  OPTION ? backup
5  OPTION ? restart
6  FIND? title quark
   TITLE SEARCH FOR...  QUARK
         105  DOCUMENT(S) ACCUMULATED.
7  ? author smith
   AUTHOR SEARCH FOR...  SMITH
           3  DOCUMENT(S) ACCUMULATED.
   ? type own

   TITLE                    A RELATIVISTIC FORMULATION OF THE QUARK MODEL FOR MESONS
   AUTHOR                   Llewellyn Smith, C.H.

   TITLE                    A RELATIVISTIC QUARK MODEL FOR MESONS AND THE 'WEISSKOPF-VAN ROYEN' PARADOX
   AUTHOR                   Llewellyn Smith, C.H.

   TITLE                    THE BETHE-SALPETER EQUATION AND THE QUARK MODEL FOR MESONS
   AUTHOR                   Lewellyn-Smith, C.H.
```

```
 8 OPTION ? backup
 9 OPTION ? backup
10 OPTION ? and title meson
11 OPTION ? restart
12 FIND? title (quark and meson#) and author smith
   SYNTAX ERROR
   TITLE          MAY NOT BE FOLLOWED BY  (
   (              MAY NOT BE FOLLOWED BY  QUARK
   AND            MAY NOT BE FOLLOWED BY  MESON
   SEARCH RESET
13 FIND? title quark and title meson# and author smith
   TITLE SEARCH FOR...  QUARK
   TITLE SEARCH FOR...  MESON #
   AUTHOR SEARCH FOR...  SMITH
          3  DOCUMENT(S) ACCUMULATED.
   ? title bethe#
   TITLE SEARCH FOR...  BETHE #
          1  DOCUMENT(S) ACCUMULATED.
14 ? type

   TITLE                      THE BETHE-SALPETER EQUATION AND THE QUARK MODEL FOR MESONS
   AUTHOR                     Lewellyn-Smith, C.H.

   OPTION ?



          restart
15 FIND? author smith after 1969 1969
   AUTHOR SEARCH FOR...  SMITH
   SYNTAX ERROR
   WORD           MAY NOT BE FOLLOWED BY  AFTER
   SEARCH RESET
   FIND? author smith date after jul 1969
   AUTHOR SEARCH FOR...  SMITH
   DATE SEARCH AFTER  AUG- 1-1969
          6  DOCUMENT(S) ACCUMULATED.
```

A.4 CALIFORNIAN DEPARTMENT OF MOTOR VEHICLES

Address: DMV, Sacremento, California.

Activities

The department is a part of the Californian State Administration. It maintains records on drivers licences and vehicles registration. Both of these files are used by a number of agencies, principally police departments, and the information is available to the general public at a fee of $.50 per record.

Systems discussed

The AMIS system provides for real-time enquiries and on-line data collection with batched up-dating to the main files maintained by the department. These are the Vehicle Registration file (19 million records) and the Drivers Licence file (11 million records). The latter includes driving offences.

The system is implemented on two RCA Spectra 70/55's and a Spectra 97/45 for communications handling. The main files are held in 23 RCA Magnetic Card Units. Each unit having a capacity of about 516 m bytes.

It is interesting to note that the system was initially justified entirely by the cost of the department's operation without reference to any improvement on the services provided. The communications system that has developed around AMIS includes about 200 remote teletypes for file interrogation by police and other agencies and several other computer systems are linked to the AMIS system. The network will be rationalised by the introduction of a single message switching computer in the near future.

Data collection is done on about 200 on-line video terminals. The data collection system is independent of the file management system and the data entries are checked for format and consistency and stored on magnetic tape for overnight entry to the main files. At present, about 100,000 entries are made each day, requiring 6 hours for file updating.

A.5 BOEING SCIENTIFIC RESEARCH LABORATORIES

Address: Seattle, Washington, USA

Activities

The information Sciences Laboratory of which Dr. Bayer is a member, carries out research in areas of company interest and provides assistence to other departments of the company on a consulting basis. Mr. Milliman belongs to a software group called AMPE currently supporting two data base applications.

Systems Discussed

References were made to the following Boeing systems:-

APL	augmented parts list
APR	magnetic tape oriented engineering design system
PDS	preliminary design system using disks and implemented using IMS (IBM software)
ECDR	disk oriented engineering configuration system

"Manufacturing" and "Purchasing"

but information was not available on all of these.

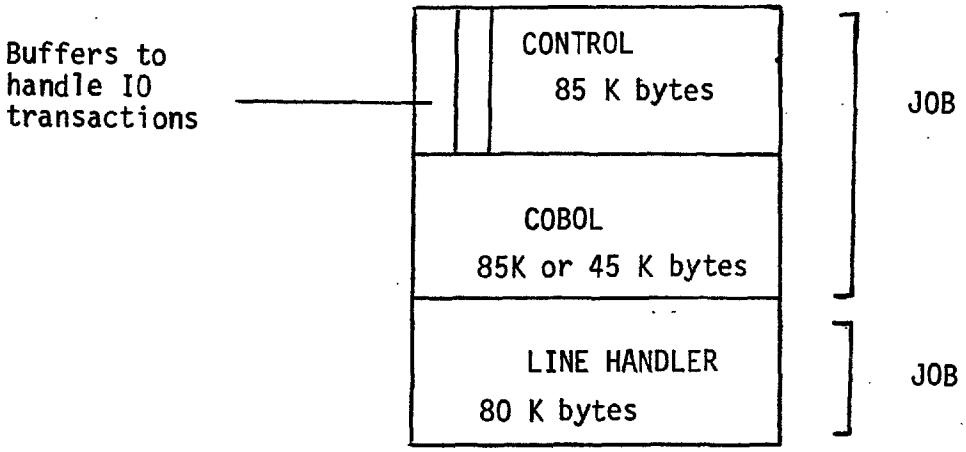
The ECDR system which is an aircraft configuration and design system was initially designed for military projects, but has since been generalised. The data base has 50×10^9 bits and uses two IBM 360's but is not on-line at present. Eventually, the data base will reside on one large 360 configuration and will be on-line for part of the working day, the system will be implemented using the same control logic as for the "Manufacturing" system discussed later.

The AMPE group has two data base applications each of about 10^9 bits; "Purchasing" including invoicing and "Manufacturing" which includes production scheduling.

The "manufacturing" system was discussed in some detail, the system has two C.P.U's, the data base resides on disk storage and magnetic tape is used to queue updates to this data base. Transactions are split into inquiries and updates, the inquiries are processed directly using the data base on one C.P.U.,

updates go to the other C.P.U. where they are stored on magnetic tape. Inquiries and the updates from magnetic tape are processed in parallel, no protection is provided to prevent possible clashes since the probability is considered very small. Transactions can contain a record disk address, a feature which has resulted in errors in the past. The C.P.U. which handles the real time inquiries and terminals has specially written software.

Core Map of Real Time Software



The system runs under OS/360 as two jobs. The control software is written in 360 Assembler because high level languages would not allow optimisation of I/O. The COBOL program can occupy 85 K or 45 K bytes depending on the C.P.U. used. The line handler runs as a separate job and is written using BTAM ("basic terminal access method" supplied by IBM) occupying 80 K bytes of core.

The control software contains a "File Handler" and a "Message Handler"; the former is responsible for insertions, deletions and retrieval, the latter communicates with the COBOL program providing messages on a sequential basis. There are two versions of the file handler corresponding to two different file structures (each file is accessed by only one key).

1) Direct Access File

In most cases the records are accessed directly since the disk address is a parameter of the transaction, a minority of transactions use a hash-coded table for record access.

2) Hash-coded Index Tables

Record access is always by hash-coded tables using either of two methods:

(a) the "hashing" algorithm generates the address of a table entry which then gives the record address.

(b) the "hashing" algorithm gives the record address directly.

Inverted file structures were considered but rejected as being too inefficient on current hardware. The CAS system was thought to be useful, particularly from the point of view of flexibility and freedom from complex data management. Using a "pointer" data base structure, the maintenance of pointers was considered a large overhead. Dr. Bayer made the point that the CAS may be inefficient for problems where long "reference chains" were involved, a parts explosion problem may involve 20 levels of reference.

A.6 BELL TELEPHONE LABORATORIES

Address: Holmdel, New Jersey.

Activities

Bell Telephones intend to integrate their business data processing systems using third generation equipment and direct access storage, and are currently assessing hardware and data management systems with a view to this integration. Mr. J. M. Wier has been responsible for implementing sophisticated information retrieval systems for use in particular problem areas.

Systems discussed

Bell Telephones have a vast data processing requirement, a large proportion of which is the administering of customer accounts. (In the New York area alone there are 5 million accounts). We discussed the requirements which the integrated systems would have to meet.

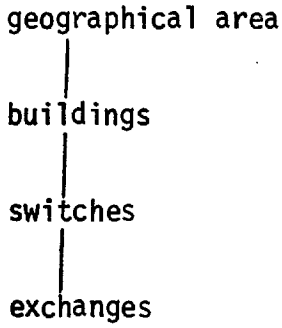
Telephone accounts will be broken down into areas each having a single data base, the practical upper limit on data base size is 10^{10} bits with present equipment. Three computer manufacturers are being considered: Univac with the 1100, IBM with a 360/65 and RCA with the Spectra 70/45, each using disk storage. Univac provides a data management scheme similar to that specified in the CODASYL DBTG October 1969 report; IBM provides the IMS system and RCA provides the IPS system. Equipment will probably be ordered from all three manufacturers.

The data bases will provide real time response for inquiries and updates together with batch processing facilities for daily, weekly and monthly sorted output. The transaction rate varies considerably reaching a peak of 2,200 in a half hour period, 700 of which are updates. Response time requirements have a spectrum of values, the lowest being several seconds, for real time inquiries. Inverted file structures were rejected because the large volume of updates would result in too much directory maintenance, the TDMS system was rejected mainly on this

count. The data base will be constructed from records and pointer systems incorporating a variety of data structures. A disadvantage of this method is that after system implementation, any new types of access path would require a major reorganisation. Updating requirements for the data bases are complex and some doubts were expressed about the adequacy of proposed systems in this respect.

Lockout is necessary at the record level to prevent clashes between updates (a reason why TDMS could not be used), however, a complete update transaction may take minutes so the locked out record must be available for reading. An important requirement is the facility to "undo" a particular transaction restoring the original value of the record. Comprehensive data validation and checking is required on transactions. Mr. Canaday emphasised that his department were still in the design and assessment stage, so some questions were still unresolved; he expressed interest in the proposed CAS system and if hardware and system were available,it would be considered as a possible alternative to the systems mentioned.

Mr. Wier is developing information retrieval systems using data bases of 10^6 - 10^8 bits. Historically, the first data base application was an inventory of local switching equipment. The data is essentially hierarchical in nature:



The equipment was housed in 8500 buildings dealing with a total of 50 million telephone lines and the data base was implemented on a G.E. 605 with direct access storage. The basic principles used in the construction of the original information retrieval system have been used in more recent system designs implemented on a IBM 360/67. Fortran was the only available language at the time so an augmented version was used to implement the retrieval system. The data base generation routines are file driven and use compiler-compiler techniques. The total size of software is about 15 million characters but much of this is redundant.

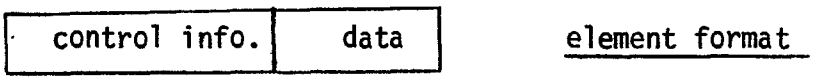
The system is fully conversational and has comprehensive diagnostic and monitoring facilities which provide statistics about system usage. The language

facilities are a subset of natural language and commands are formed from verb clauses. (It is found that about six different clause types are adequate). Examples of verbs are:-

DISTRIBUTE (provides a statistical distribution)
TOTAL
EXTRACT

There is also a verb clause used for curve fitting. Data is organised into a fieldwise structure where it is stored by property rather than by record. There are 55 basic properties augmented by functions allowing the user to express more complex entities as a combination of the basic properties. This organisation provides good response for statistical queries, a simple statistical request can be answered in less than 5 seconds. The system can also handle queries about records rather than properties, but its efficiency in this case is low.

The data base consists of two files, the retrieval file and the update file. The retrieval file has a fieldwise record structure[1] with data organised into 55 properties. Each property corresponds with a vector of typically 15,000 elements.



The control information associated with each data item allows records to be reconstructed from the property vectors. Within a particular property vector the element length is fixed; typical sizes for the data fields are 4 bytes for numbers and 20 bytes for alphanumeric information. In addition to the property vectors, there are directories describing elements and master records containing structure information. The update file has a record-oriented structure since this is the form in which data enters the system. Updates are accumulated in this file and periodically the retrieval file is updated by fragmenting the records into their constituent fields. This method is adequate since the update volume is low.

1 This is an inverted file organisation.

A.7 NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Address: NASA, Scientific and Technical Information Division, Washington, D.C.

Activities

This divsion of NASA is responsible for the maintenance and accessing of a bibliographic file on all technical documents relevant to NASA's operation; this includes published articles, technical reports, project records, and correspondence. The current file contains 600,000 document records and is growing at the rate of 8000 per month.

Systems Discussed

The prinicipal system used is called NASA/RECON. This provides sophisticated on-line facilities via 22 video terminals for information retrieval from the bibliographic file. The system is implemented on an IBM 360/50 with 512 byte core, two 2314 disk drives (233 M bytes, 75 ms average access) and two Data Cell systems (400 M bytes, 350-550 M sec. average access). The main file (stored in the Data Cells) is accessed via six indexes (stored in the 2314's) ordered by: (i) authors, (ii) subject keys, (iii) source, (iv) contract number, (v) accession number, (vi) report number. Facilities are provided for users to list documents with any combination of properties in any of these categories. The lists generated may be displayed or used for further retrieval by the generation of a more restricted list.

A parallel system called STEMS is used for file updating and report generation in batch mode. Updating consists only of file extension since records are very rarely deleted. Garbage collection and file re-organisation occurs about once a year.

Record structure is independent of the problems in the system since each type of record is accessed only via a corresponding 'descriptor table' as in the CRH system for Basic Language. High level facilities for record structure definition are not provided because this is an infrequent operation, the desirability of such facilities in general was acknowledged.

The use of content-addressed store for index information was considered very attractive as a means to reduce processor time spent in index searches. Some discussions had taken place concerning a "Rapid Search Machine" developed and marketed by the General Electric Company for this purpose using a modified magnetic tape system.

A.8 University of Chicago

A simple library information retrieval system was discussed at the University of Chicago. Details of the system were not provided apart from answers to the questionnaire, these answers appear in 2.5.

Appendix B

Batch-Processing User Reports 1968

B.1 The Automobile Association

B.2 BP (Llandarcy)

B.3 GKN (South Wales)

B.4 Dunlop Semtex Ltd

B.1 THE AUTOMOBILE ASSOCIATION (A.A)

People Seen: DP Manager, Senior Systems Analyst

Activities

The AA is a decentralised organisation providing various services to member motorists. The main office is in London and there are 35 area offices which deal with geographical areas of Britain. The EDP department is situated in London and based on an ICT "1904 Computer" with magnetic tape backing-store acquired in June 1965. A progressive transfer of DP activity to the computer has been accomplished largely by direct automation of existing manual systems. The main system currently automated is the "membership system" which deals with the maintainance of membership information and associated transactions. The current complex payroll system is not mechanised since it is of an unsuitable structure having evolved over a long period of time. Card files are held at the area offices duplicating information held in the computer. The full benefits of automation are still to be realised by a policy of centralisation and overhaul of outdated systems together with an upgrading of the 1904 with disks and remote CRT terminals.

The Membership System

The membership system is built around the membership master-file which occupies in excess of thirty reels of magnetic tape. Because of the decentralised nature of the AA's administration the master file is separated into 35 sub-files each with its own header record and trailer record containing summary information. Examples of the formats of area headers and membership master records may be found in Fig. B.3 and Fig. B.4 respectively. Management enquiry facilities are built into the system (but are not shown in the discussion below) allowing management to select certain pre-defined interrogations which are performed during the normal operation of the membership system.

A simplified schematic of the daily operation of the membership system appears in Fig. B.1. A reduced copy of the original more detailed schematic appears in Fig. B.2 and has been marked to show its relation to the simpler version. Referring to Fig. B.1 , essentially there is a build-up of information until stage D where the data is filtered into the different subsystems which provide

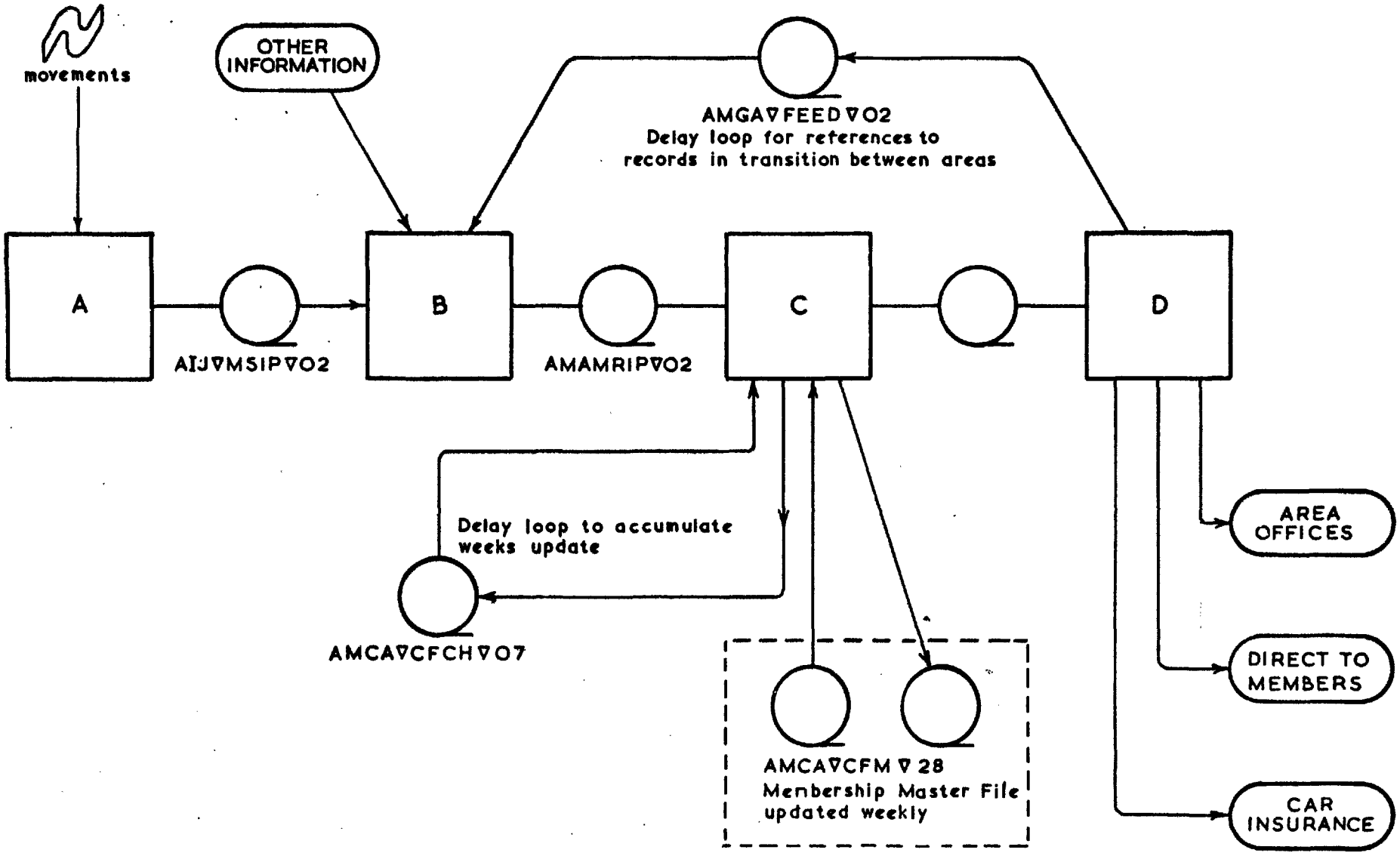


Fig. B.1 Membership System

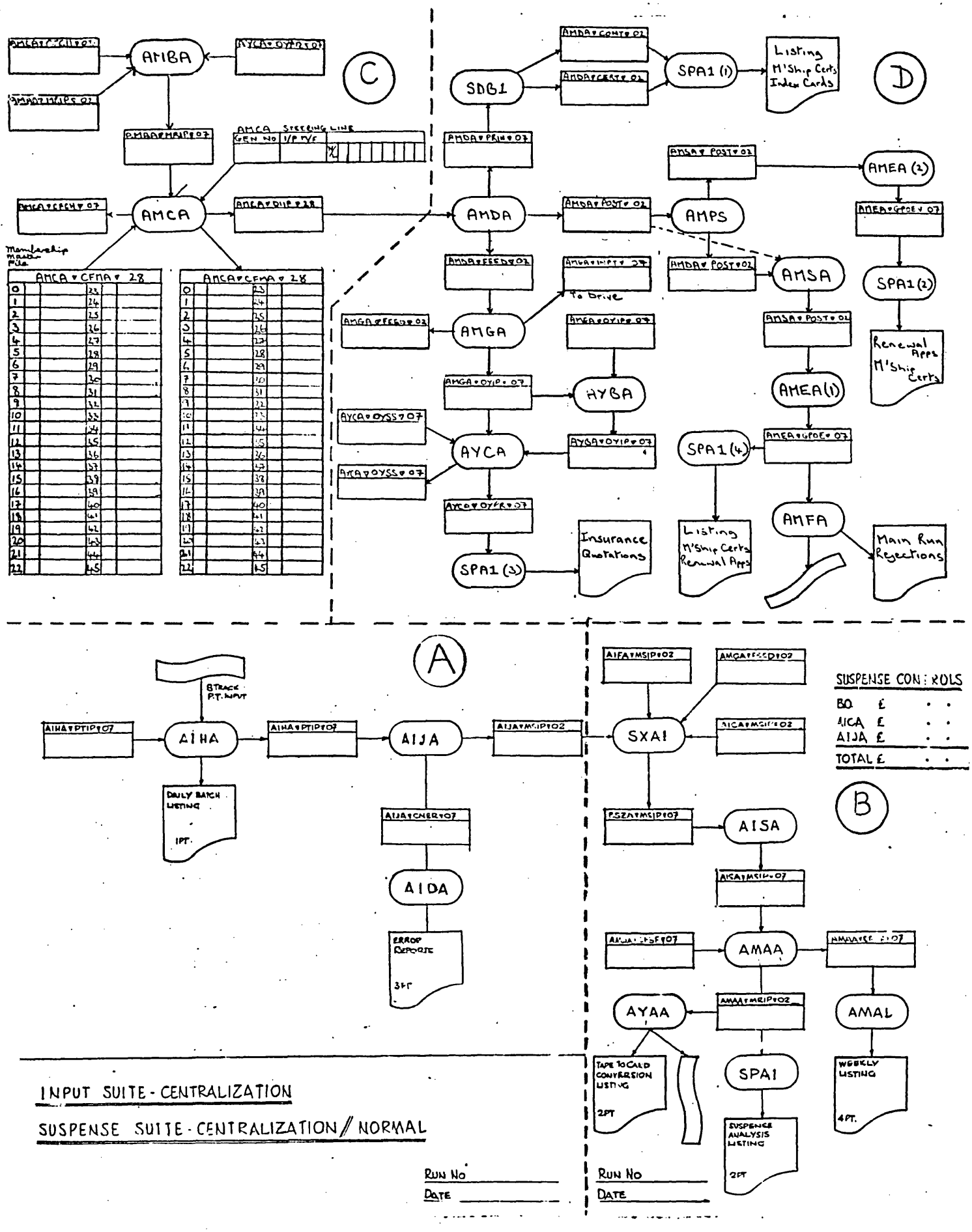
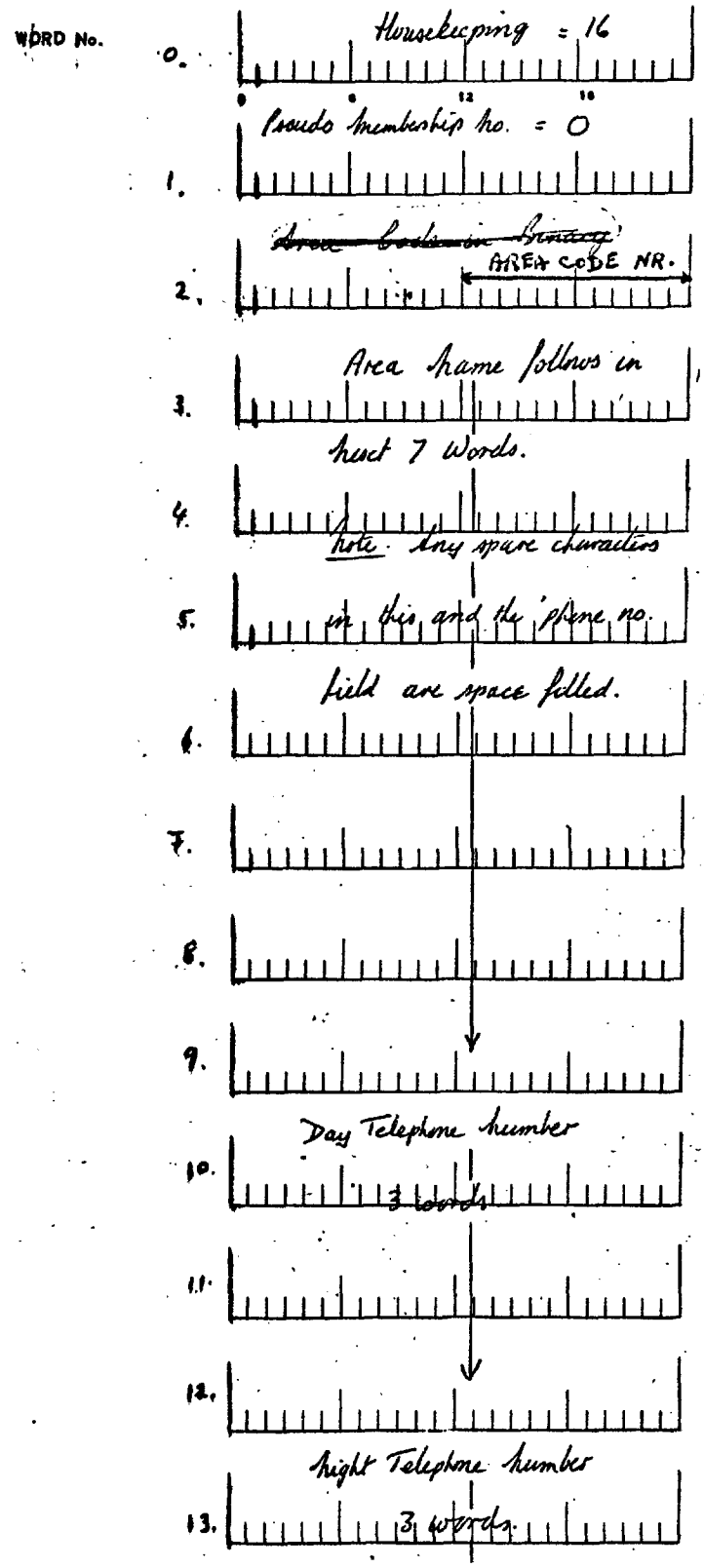


Fig. B.2 Original Documentation for Membership System

SECTION: 2.7.1.2	RECORD LAYOUT SHEET				
PROGRAMMER	DATE	MEDIUM	PROGRAM	PAGE	FILE NAME
	/ /	MT	AMCA		AMCA CAMA 28
RECORD TYPE					* INPUT TO AFAA
AREA HEADER					* OUTPUT BY AMCA

* DELETE IF INAPPLICABLE

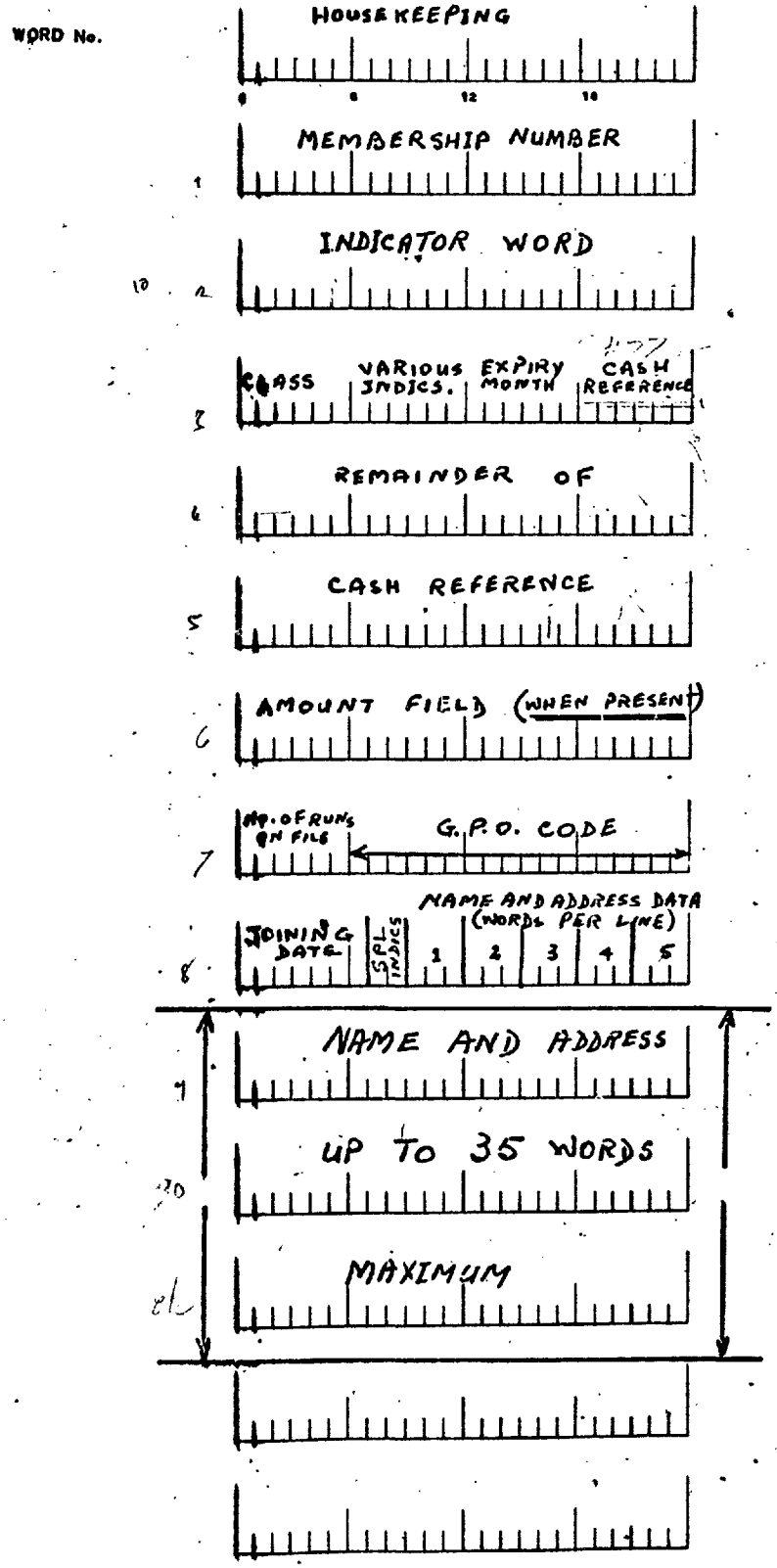


NB All fields in binary except where notified otherwise

Fig. B.3 †Area Header

SECTION:		RECORD LAYOUT SHEET			AA
PROGRAMMER	DATE	MEDIUM	PROGRAM	PAGE	FILE NAME
	/ /	MT	AMCA		AMCA CFMA 28
RECORD TYPE					* INPUT TO AFAA
C/F. MEMBERSHIP MASTER RECORD					* OUTPUT BY AMCA

* DELETE IF INAPPLICABLE



NB All fields in binary except where notified otherwise

Fig. B.4 Membership Master Record

information (i) to Area Offices, (ii) direct to members, (iii) for administration of Car Insurance. Paper-tape input is batched to magnetic tape at stage A where batch totals are checked and data validation performed. At stage B other records are added to produce a new file containing about thirty different record types. This file in stage C is then run against the master file to extract subsidiary data, check transactions and accumulate changes for the weekly update (not shown) of the master file. Stage D performs the separation of transactions for each of the subsystems. A delay loop exists from D to A holding transactions for members transferring between areas - a process which may take two or more master file update runs. These transactions are delayed until the transfer is complete.

System Changes

Each program selects only the record types it recognises from the different files. Additions of new record classes can be accommodated without disturbing existing programs, the Car Insurance subsystem was added easily because it required completely new record types. A recent instance was recalled of a format change in the master file. The "cash reference" field see Figure B.4 was extended by one word. Alterations had to be made to the following programs:-

AMAA	AMDA	AMEA
AIJA	AMGA	+5 enquiry programs
AYAA	AMBA	
AMCA	AMPS	

Of the above programs only 3 actually used this cash reference field the others were affected purely by the reformatting.

B.2 BP (Llandarcy)

People seen: D.P. Manager, Senior Systems Analyst

Activities

BP (Llandarcy) is an oil refinery owned by BP International and manufactures petrol, aviation spirit, diesel oil and fuel oil as the main products. Similar refineries exist in other parts of the BP group at Grangemouth and Kent. Each of these refineries has its own 1900 series computer and the D.P. activities are organised in a similar way. BP Llandarcy have a "1904" with magnetic tape and disk secondary storages, this replaced an "ICT1301". The current DP activity is entirely batch-processed and is an evolution of the original system on the 1301 which had become overloaded. All master-files are held on magnetic tape, the disk is used mainly for short duration transaction files and shows advantages in a reduced sorting time for the latter. There are three main DP activities:-

(i) Stores System

(ii) Management Accounting System

(iii) Payroll system.

Marketing and distribution of the refineries' products is performed by a separate company "Shell-Mex and BP" set up purely for this function. Consequently, sales ledger and invoicing activities are not performed at Llandarcy. The three D.P. activities mentioned above are well established and there are no requirements at the present time for major evolution of the system.

Stores System

This is an integrated system for inventory control based on magnetic tape files. The system is composed of three main parts:-

Ordering: issues orders for goods and materials required

Accounting: maintains the value of stock items held and inputs information to the Management Accounting System

Stock Control: maintains the quantities of physical stock and re-order levels for the issue of orders.

The day's stock movements are used to update the "Main Stock File" which (see Fig. B.5) holds both the value and physical information about items in stock. A subsidiary stock file is used to hold certain details of ordered goods and invoices received. This file is used because cash flow is not in step with the transfer of goods. The sotck levels on the Main Stock File are checked against an exponentially smoothed expected demand, if the stock in hand is unlikely to meet the demand an order is issued so that taking account of "lead time" the stocks will be replenished before danger level. If a stock item is almost at danger level but an order has already been placed then a "speed-up order" is issued. The quantity ordered is optimised with respect to cost and storage capacity. During the daily Stores System runs, the transactions are accumulated on a Monthly Transaction file which is used for costing purposes in the monthly generation of Management accounts. This file represents the interface between the Stores and Management Accounting Systems.

Management Accounting System

The management accounting system exists mainly for the production of three ledgers:-

(i) The Operating Ledger:- containing the detailed operating costs for the refinery.

(ii) The Capital Ledger:- containing details of the budget and job expenditure.

(iii) The Maintainance Ledger:- containing detailed costs of maintainance.

The plant is broken down into cost centres which generally include details of personnel costs, chemical costs, works fuel and services. The ledgers contain figures for both estimated and actual costs.

Fig. B.6 shows how the production of the ledgers is broken down into six "unit systems". The input from other systems occurs is indicated by A, primarily this comes from the Stores System and the Payroll System. Card input is batched to tape and validated before

APPLICATION B. P. REFINERIES INVENTORY CONTROL - MAIN STOCK FILE DATE

SECTION HEADING RECORD

	DATA	WORD COUNT	SECT. NO.	VOCAB. NO.	TYPE					ORDER COST	SECTION TITLE
		BINARY	▽NN	000000	1	▽▽	▽NN	NN	NNN	DEC. PENCE	XXXX

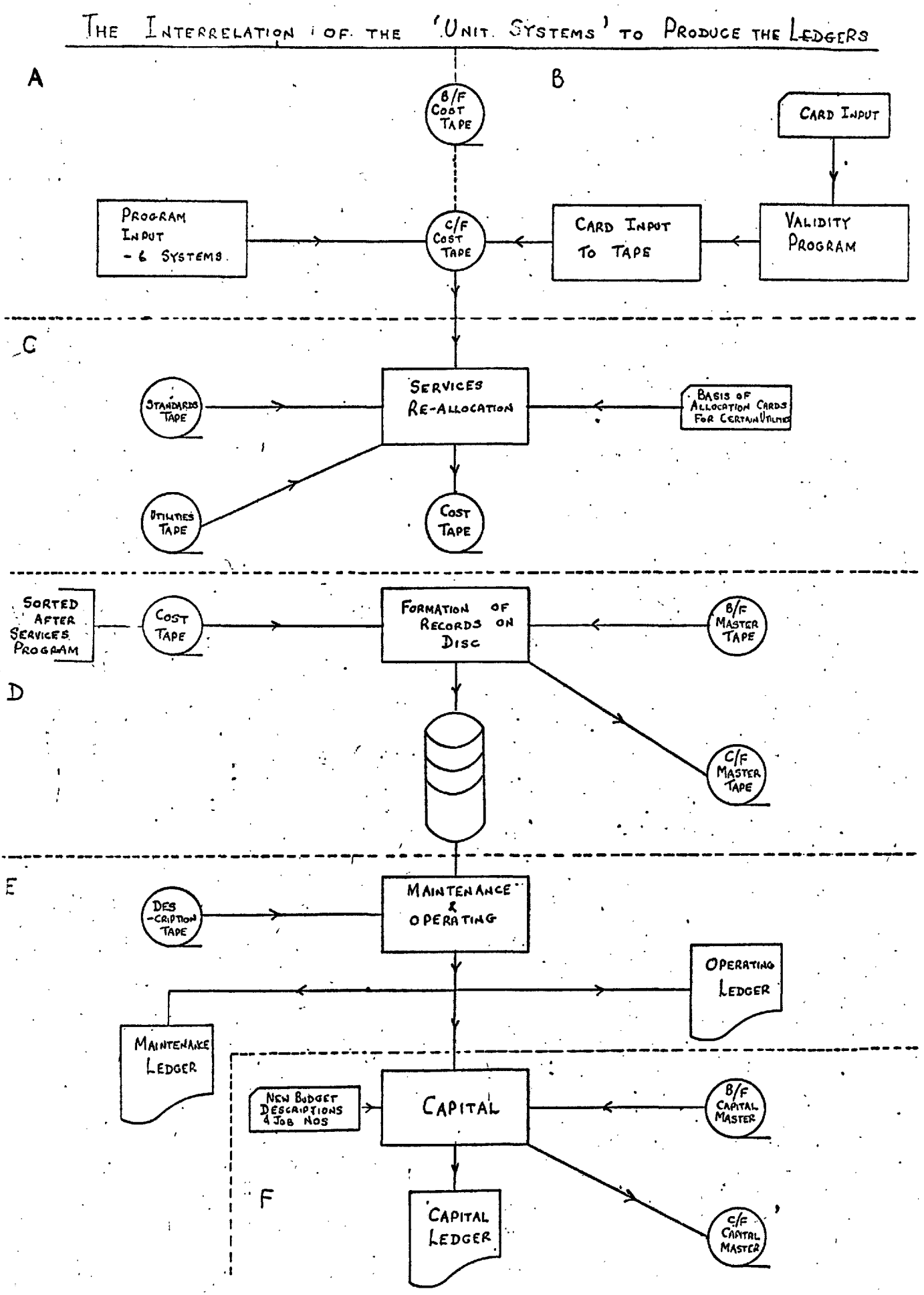
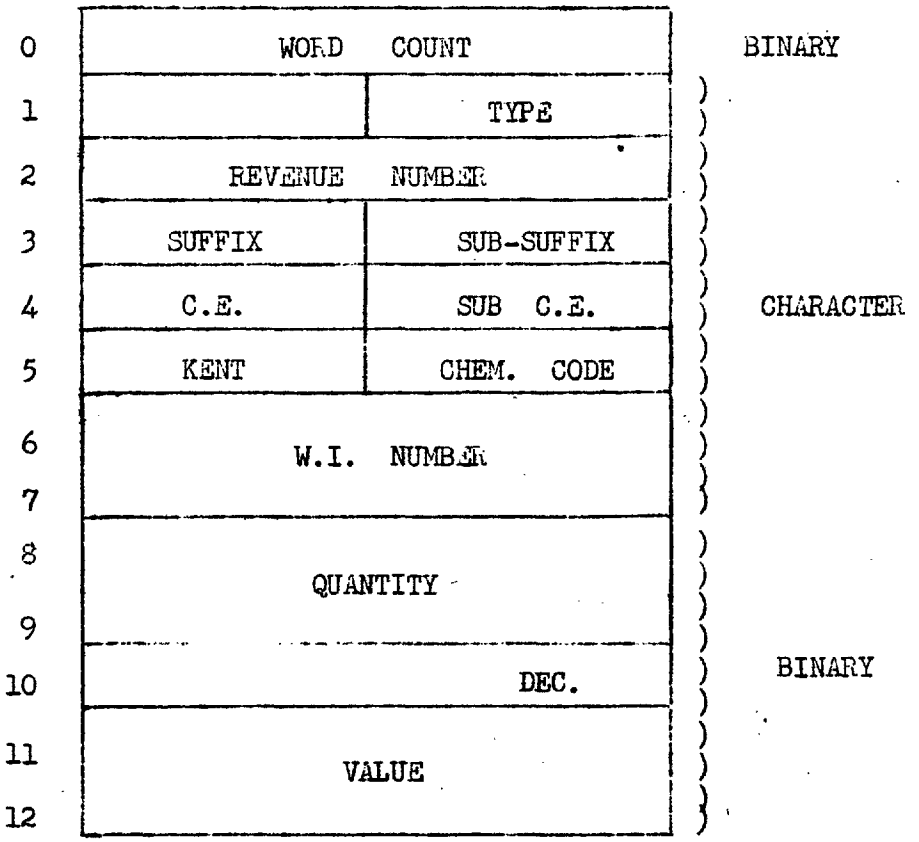


Fig. B.6 Management Accounting System

MAINTENANCE, OPERATING & CAPITAL LEDGERS

TAPE RECORD FORMAT

COST DETAIL TAPE



RECORD TYPES

i) Cost Detail.
ii) Variable Estimates.
iii) Input Tons.
iv) Output Tons.
v) Losses.
vi) Stream Hours.
vii) Non Standard Estimates
viii) Steam quantity as per Flexible Budget.

Fig. B.7 Example Record Format

being merged with the A input to produce one tape containing costs for the month. Unit system C adds information about services to this cost tape which is then sorted. Disk records are formed containing the basic information for the three ledgers and the cost master tape is updated in D. Unit system E produces the Maintainance and Operating Ledgers from the disk records expanded by alphanumeric descriptive information from the "Description tape". Finally the Capital Ledger is produced by F which also updates the Capital Master. An example of a record format common to all the master files is a cost detail (see Fig. B.7).

Payroll System

The Payroll System has three separate "runs" resulting finally in the production of payslips, credit transfers, cost listings and various analyses. The separate runs are shown in Fig. B.8. Run I is an input and validation run from cards to disk or tape. The input is then sorted and personnel identity number checked. Run II takes the sorted input data from disk/tape with the Salaries Master file to produce a series of disk records per man for payslip, credit transfer and cost allocation. During this run the salaries Master file is updated. The file is now sorted so that all payslip records are contiguous and appropriately sequenced, similarly for the other record types - pre-printed stationary being used later. Run III now prints out payslips, credit transfers and cost analyses, during this process information is added to the "months cost tape" to provide input from salaries into the Management Accounting System.

System Changes

The fields in a cost record were altered. Four programs required amendment as a result, three of these programs did not use the modified information but were affected by the reformatting.

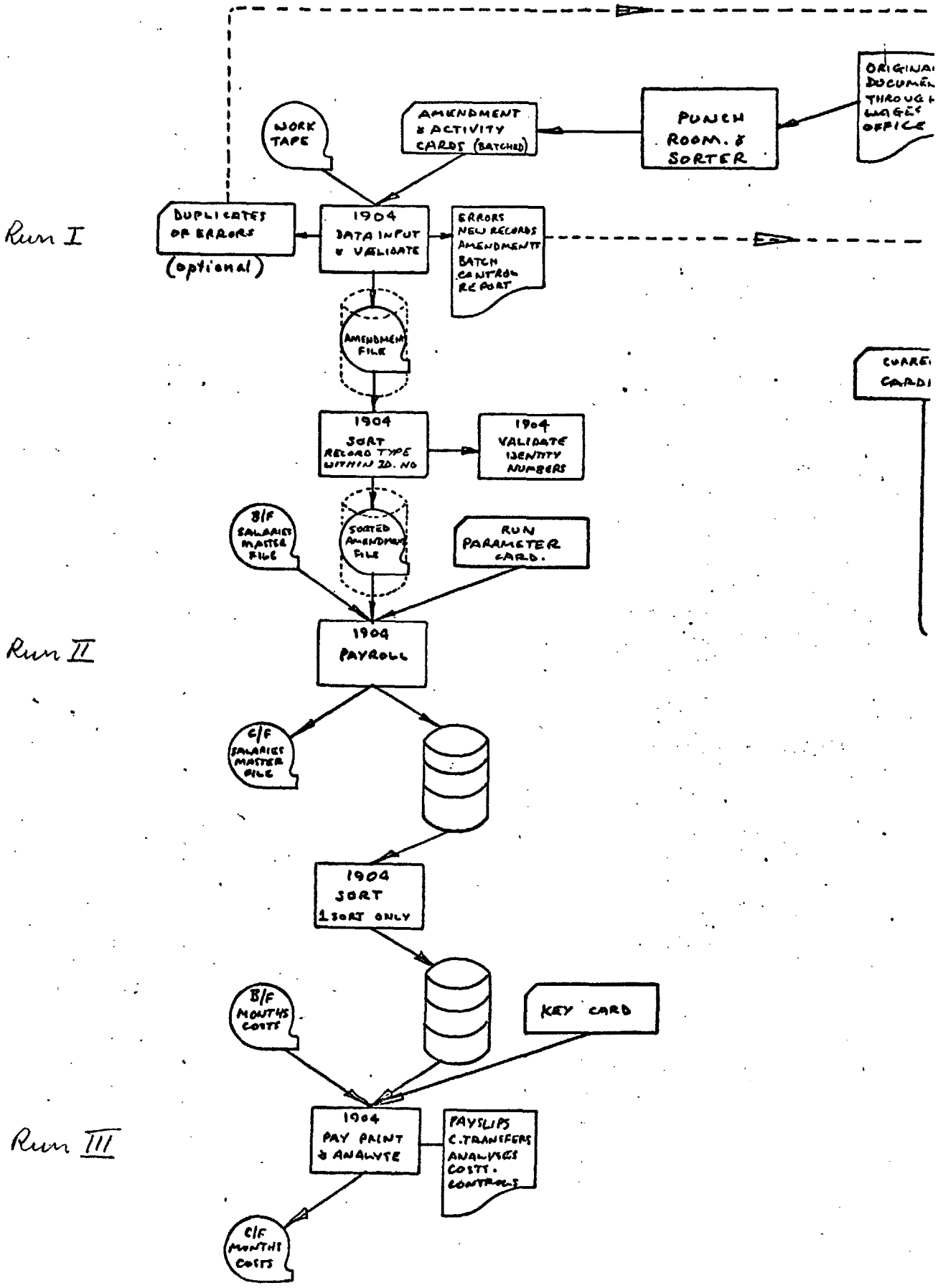


Fig. B.8 Payroll System

B.3 GKN (South Wales)

People seen: Senior Systems Analyst

Activities

GKN (South Wales) is a constituent company of GKN Rolled and Bright Steel Ltd., the latter being an administrative sub group within the GKN group. GKN (South Wales) consists of five separate factory units in the same locality, based on five distinct product groups which include reinforced bars, nails, coil and wire. The commercial and financial aspects of the company combine to form a central hub which oversees these five factories. The computer system is a 1902 with magnetic tape backing store and a variety of slow peripherals. This machine replaced a Powers-Samas punched card installation and the computer systems are largely derivatives of their punched card predecessors. The EDP department is under the control of the Financial Director and the main applications connected with accounting, invoicing and payroll. Order and sales analyses are also performed and although records of stocks (both raw material and finished goods) are kept these are intended to aid the accounting function, no automated stock control system exists.

An interesting point which emerged from the discussions was that GKN have adopted a method of programming which permits certain degrees of format-independence. Program record areas are addressed indirectly via a table of pointers. A table comprises a simple vector of indirect addresses and there is one table per record class. The indirect addresses can be altered after loading a program and prior to file processing, allowing different record formats to be accommodated. This system suffers from the following disadvantages when compared with the CRH system (described in Chapter 4):-

(i) The record structure is a simple linear list of fields.

(ii) High-level record description (schema) facilities are not used.

(iii) The addressing mechanism is insecure since the table itself is accessible by the program, furthermore there is no bound-checking on field limits.

(iv) The alteration to indirect addresses to achieve format-independence is a completely manual procedure and insecure in comparison with the self-describing files and automatic binding of the CRH system.

The Billet Stock System

The system for "Billet Stocks" (i.e. stock of raw material) was considered representative and was discussed in more detail. Fig. B.10 gives the flow of the daily update run for the Billet stock master file. File formats appear in Fig. B.9, the linear record structure is apparent. A brief description of the diagram of Fig. B.10 appears below:-

Program Run	Description
BS02	Validity check paper tape input print control totals and write to movements M/T.
SBS1	Sort movements tape.
BS03	Run the sorted movements against Billet Stock master file for further checks and expand movements with alphanumeric information.
BS04	Update the Billet Stock master and produce the daily billet movements tape by minor modifications (see DAILYBILMVTS of Fig. B.9 and CORRECTBSTAMS)
MBS1	Merge the daily billet movements
SB03	Process merged tape to produce weekly movements

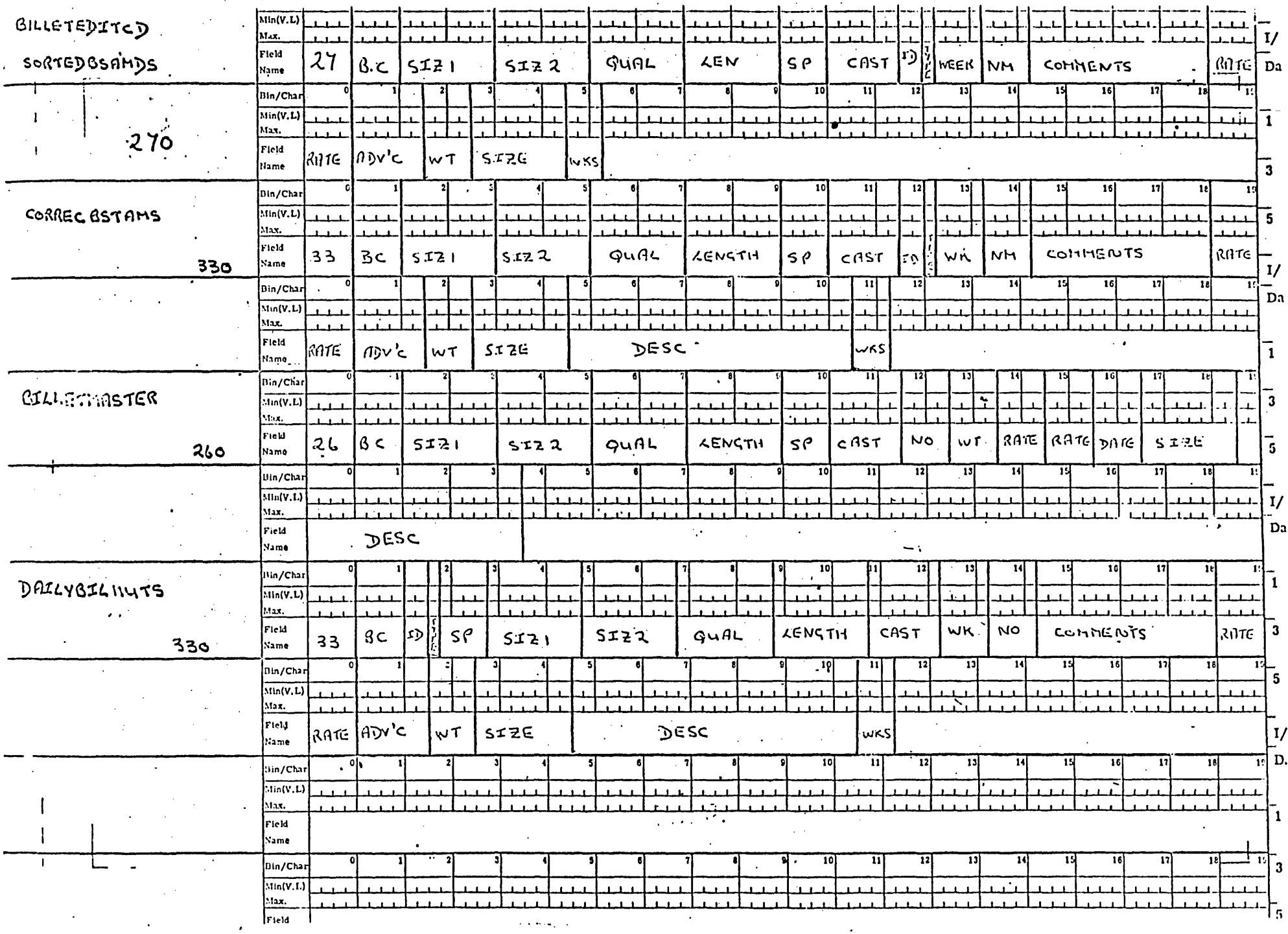


Fig. B.9 Billet Stock File Formats

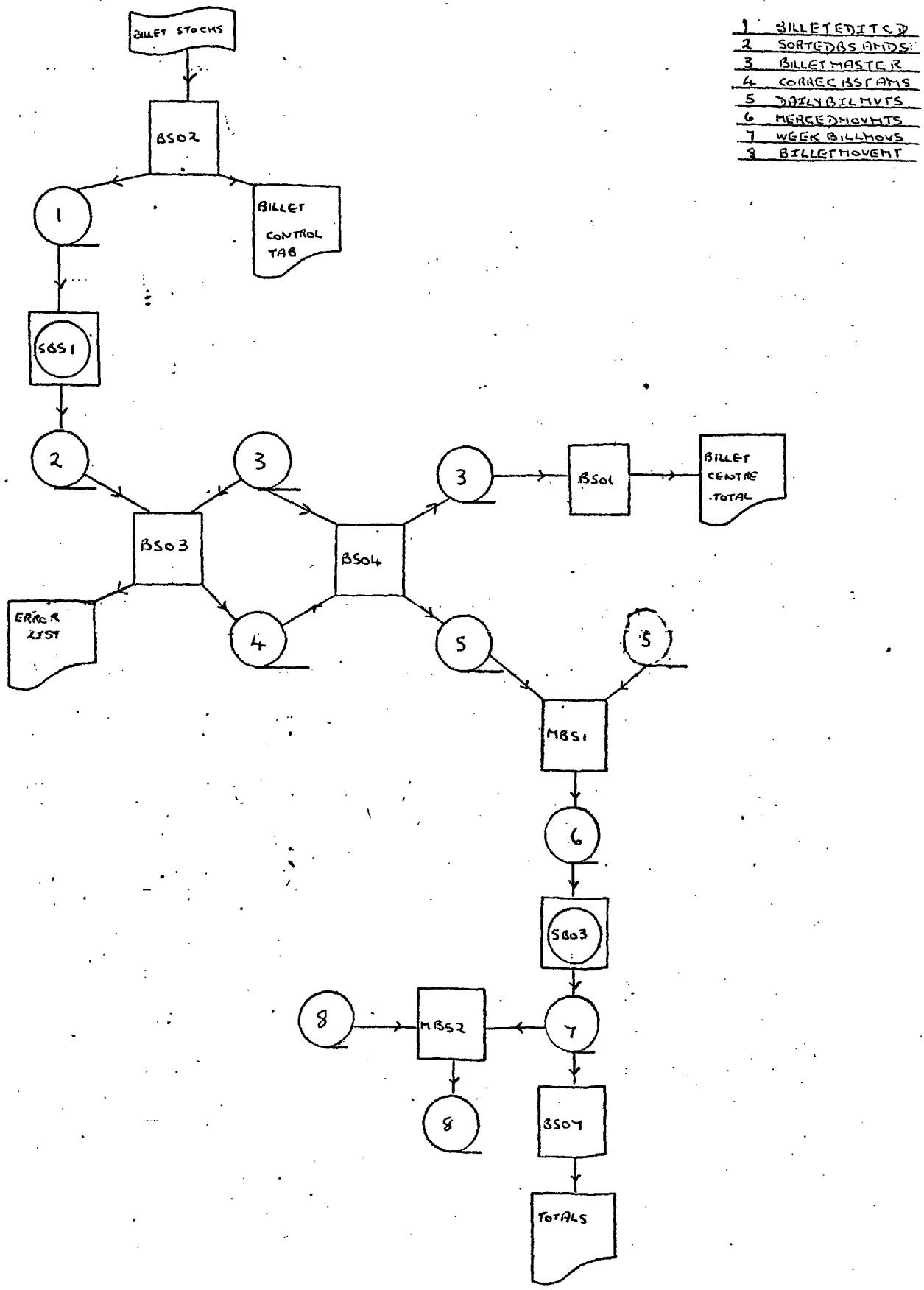


Fig. B.10 Billet Stocks System

Program Run		Description
MBS2	weekly runs	Update cumulative movements.
BS07		list weekly movements.

System Changes

There had been minor format changes to the files but these had been successfully accommodated by the indirect addressing scheme mentioned earlier.

B.4 DUNLOP SEMTEX LTD.

People seen:- D.P. Manager, Systems Analyst

Activities

Dunlop Semtex is part of the consumer group of the main Dunlop Organisation. They manufacture floor covering such as tiles and vinolay supplying these to the trade via depots on a wholesale basis. This is one half of the company's business the other half is concerned with contract work undertaking the laying and fitting of the company's products on fairly major building undertakings. Dunlop Semtex are awaiting (at the time of this survey) the delivery of a 1902A with magnetic tape and disk secondary storage.

Systems Discussed

At the time of conducting the survey the application program suites were being written in anticipation of the machine delivery. The main pressure was that of short timescales allowed for the implementation so COBOL was being used exclusively. However, the point was made that the compilation was slower than anticipated and object programs were considerably larger than equivalent assembler programs.

The main application area is concerned with invoicing and associated activities and the switch to computer from manual methods was justified largely on substantial tangible savings in invoicing. Dunlop have a complex tiered structure of discounts and the manipulation of these together with more efficient control of debtors will be of major benefit to the company. Payroll will also be automated and a management information system will eventually be developed based on the accumulated invoicing information.

The systems were not discussed in further detail due to the implementation pressures on staff availability for discussions, however answere were kindly provided to the questionnaire in spite of these pressures.

System Changes

Although the program suites were still being specified a system change had occurred which required re-programming. Four new fields were added into a master file record and this gave rise to modifications to three programs.

Appendix C

Detailed ECAL Language Description

C.1 Descriptive Notation

C.2 Store Allocation and Computational Commands

C.3 Task Commands

C.4 Data Mapping Commands

C.5 Stream and Program Oriented Commands

C.1 Descriptive Notation

ECAL is a context dependent language. The single syntactic class "name" (or identifier) is subdivided into nine classes (see later), according to the context only certain of the classes are allowed. A convenient notation for describing the commands of the language is a modified form of BNF where we specify not only a "name" but indicate to which class the name belongs.

<record name>	indicates a name which has previously been declared as belonging to the record class.
<record name>	indicates that the name is declared to belong to the class "record".

We will illustrate the notation using two commands:-

RECORD <record name> FIELDS <LIST <field name>>;

The above command initialises the names which occur in the specified context to belong to the record and field classes respectively.

DESCRIBE <record name>;

The above command specifies that only a name which has been initialised as in the record class is valid.

The following sequence is therefore valid:-

```
RECORD MASTER FIELDS MAN, HOURS, AGE;
DESCRIBE MASTER ;
```

Character Set

<X> alphabetic characters (upper case)
<n> numeric characters
+ - / * & ↑ ; , ( ) [ ] ' £ ← = ξ : " % @

'blank' is significant only when acting as a terminator for names.

Basic Syntactic Types

<name>	:=	<x>| <name> <x| n>
<integer>	:=	<n> |<integer> <n>
<octal>	:=	% <integer> not including 8,9
<CHS>	:=	character list consisting of any character
<CHS not C_1C_2..>	:=	character list consisting of any character except C_1C_2...
<literal>	:=	"<CHS not"> "
<op>	:=	+ - / & * ↑
<unop>	:=	+ -
<LIST <item>>	:=	a list of <item> separated by ,
<comp>	:=	EQ, NE, GT, LT, GE, LE (comparison operators)
<logic>	:=	&, ↑
{<element>}	:=	indicates the optional occurrence of element

Labels

<label> := <label name> :

Every command may be preceded by a label (i.e. labels are valid in stream and program modes) so they will not be included in the command descriptions of the following sections.

Textual Comment

A comment begins with @ and terminates with a new line.

Object Classes

There are nine classes of object, each programmer-defined name belongs to one of these classes.

VARIABLE:
This is an array consisting of one or more C.P.U. "words" containing positive or negative integers.

STRING:
This is an array consisting of one or more character strings. Each character string in the vector may vary up to the same maximum.

BUFFER:
This is a vector of words used to hold information transferred to or from the content-addressable store.

RECORD:
This identifies a class of "CAS records" and describes the class as a list of items.

ITEM:
This is a description of a class of "fields" which form a CAS record. In a record there may be none, one or more fields corresponding to an item. There are two classes of item "A" where the data "value" of the field corresponds to a "string" and "N" where it corresponds to a "variable".

TASK:
This is some operation by which information is transferred between the core store and CAS.

PROGRAM:
This identifies a stored sequence of commands.

FORMAL:
This is a dummy variable which is replaced by the corresponding parameter.

PARAMETER:
This is a character string which replaces a particular format.

C.2 Store Allocation and Computational Commands

Of the nine object classes only three are directly associated with storage allocation these are:-

Variables
Strings
Buffers

Variables and strings are the basic data types of the ECAL language and correspond to storage of numeric and character information respectively. Arithmetic and string expressions can be constructed from the corresponding data types and appropriate operations.

Buffers contain data which has been involved in a transfer to/from the content-addressable store which requires data in a different form to the way variables and strings are stored. The data in buffers cannot be addressed directly for computational purposes. A number of Data Mapping commands (ADD, COPY, COUNT) allow data to be transferred between Buffer storage and variables or strings for computation. Buffer storage is required for the tasks described in C.3.

Strings and variables are always elements of arrays with a fixed number of elements. An element is indexed by following the array name with a bracketed arithmetic expression. The name of the array when used alone indicates the first element, a single variable is an array of length one.

Examples: NAME is an array of 10 strings each of up to 15 characters.
ARR is an array of 20 variables.
IND has the value 4.

ARR or ARR(1) indicates the first variable in ARR.
ARR (1 + IND/2) indicates the third variable in ARR.
NAME(IND) indicates the fourth string in NAME.

Strings have an attibute "length" which gives a count of the number of characters currently in the string, the LENGTH command extracts the value "length". A substring of any string may be selected by qualifying the string name with the positions of the first and last characters in the substring (see string expression below).

BUFFER Command

BUFFER <LIST <buffer initialisation>> ;
<buffer initialisation> := <buffer name>(<integer>)

The command declares a number of buffers each of a length (in words) indicated by "integer".

STRING Command

STRING <integer><LIST <string initialisation>> ;
string initialisation := <string name>|<string name><attribute>
<attribute> := (<integer>)

This command declares arrays of strings where the attribute indicates the number of elements, the absence of an attribute indicating an array of one element. The integer which follows STRING indicates the maximum number of characters in each string introduced by the command.

Example: STRING 10 NAMES (15), PARENT;

Each string has up to ten characters.

VARIABLE Command

VARIABLE <LIST <variable initialisation>>;

<variable initialisation> := <variable name>|<variable name><attribute>
<attribute> := (<integer>)

This command declares arrays of variables, the attribute indicating the number of elements. The absence of an attribute indicates an array of one element.

Example:

VARIABLE TOTAL (20), SUM;

A.E. (Arithmetic Expression)

<variable> := <variable name>|<variable name>(<A.E.>)
<operand> := <variable>|<integer>|<octal><bracketed expression>

<bracketed expression>:= (<A.E>)

<A.E.> := <operand>|<unop><operand>|<expression> <op><operand>

Examples:

-(A+B), (A+PAY(12))/(B2+A)*G),

-HOURS , PAY(A*B/(G+A)), PAY(HOURS),

HOURS ↑(PAY & 201) , HOURS'(-2)

The operators + - * / have the usual meaning and operate on words containing either positive or negative integers. Overflow is ignored and the division operation produces an integer result. The operations &, ↑ are the logical operators .and., .or., .not., respectively. The ' operator indicates a logical shift where the sign and magnitude of the following operand gives the direction and magnitude of the shift (positive for right shift, and negative for left shift). The operator precedence is listed below, if operators of equal precedence are encountered the evaluation is left to right.

Operator	Precedence
'	1
(unary) + -	2
/ *	3
+ -	4
& ↑	5

S.E. (String Expressions)

string :- <string name>|<string name> (<A.E.>)
substring selector :- [<A.E>, <A.E>]
substring :- <string> <substring selector>
string element :- <string>|<substring>|<literal>
S.E. :- <string>|<S.E> + <string element>

Examples:

Suppose NAME contains the character sequence JIMMY
NAME 2,4 refers to the characters IMM
NAME + "JOE" refers to the characters JIMMYJOE

ARITHMETIC ASSIGNMENT Command

<arithmetic assignment> := <LIST <variable>> =<A.E> ;
Example: PAY, HOURS = (A + 256) * B

The arithmetic expressions is evaluated to give an integer which is then assigned to each of the variables on the left hand side of the assignments.

STRING ASSIGNMENT Command

<string assignment> :=<LIST <string>> <S.E> ;
Example: NAME (10) ← "PR1/00/ACF" + NAME (2) '

The string expression is evaluated to give a resulting character sequence which is then assigned to the string or list of strings on the left hand side of the assignment. The S.E provides variable length character sequences and a check is made that the strings on the left of the assignment have sufficient storage for the assignment. The "length" (see C.2) of the assigned variables is adjusted to equal the number of characters being assigned.

LENGTH Command

LENGTH <string> INTO <variable> ;
a count of the number of characters in the string is inserted in the specified variable.

MSTRING Command (make string)

MSTRING <variable> ,<string> ;
This command copies the contents of a variable bit for bit into a string. The string will have a length (see LENGTH Command) of four after this operation.
Example: MSTR NUMBER (2), NAME ;

MVAR Command (Make Variable)

MVAR <string>, <variable>;
This command copies the string bit for bit into the variable. If the string is not of length four (see LENGTH Command) there will be implementation dependent rules for performing the copy operation.
Example: MVAR, NAME, NUMBER ;

C.3 Task Commands

The ECAL Task Commands are used for the following operations:-

	Command
Retrieval by content-addressing :	SEARCH
Insertion of new record :	OUTPUT, WRITE
Deletion of an existing record :	DELETE
Abolish a Task freeing the buffers:	CLEAR

The SEARCH command provides full content-addressing facilities and results in the execution of a single "CAS Controller search task" (see 3.2.4). The OUTPUT, WRITE and DELETE commands involve store management and hence the direct access facilities of the CAS Controller mentioned in 3.2.1. The CLEAR command does not involve the CAS Controller it results merely in actions within the ECAL software.

SEARCH Command

SEARCH <task name> FOR <record name> PUT {ALL} <retriever> INTO <LIST <buffer names>> IF <selector> COUNT <variable> ;
<retriever>:= CAL/CAL <LIST <item names>>|<LIST <item names>>
<value> := <variable>|<integer>|<octal>|<string>
<literal>
<primitive>:= <item name><comp><value>|£<item name>
<comp><item name>|<item name>
<component>:= <primitive>|<bracketed selector>
<bracketed selector> := (<selector>)
<selector> := <component>|<selector> <logic> <component>

This command defines a task and associates a list of buffers with that task. There are two variants of the command corresponding to "single-hit" and "multi-hit" tasks of the CAS controller. If the optional part is omitted then only one CAS record will be selected regardless of how many records actually correspond with the "selector", the items of the "retriever" identify which fields are to be retrieved from the selected record. If the optional "ALL" appears then the retriever will be applied to all

records satisfying the selector. Cross-coupled triples (see 3.1) are identified by the "£" symbol, also a triple may be replaced simply by an item name in which case a test for the presence of the item is performed rather than a comparison with a value.

Information retrieved from the CAS is placed in the buffers associated with the task. The retrieved information exists as a series of "results", each result deriving from one CAS record for which the selector is true. The number of results is indicated by the value returned in the variable following "COUNT" keyword. Retrieved fields cannot be directly processed in the buffers due to their variable nature. Two commands COUNT and COPY allow the data values of the buffer fields to be transferred into "variables" or "strings" for computation without the user being aware of the formatting of buffers or the distribution of information between the buffers.

Example

An example is provided in 3.3.4..

OUTPUT, WRITE Commands

OUTPUT <task name> FROM <buffer name> AS <record name>;
WRITE <task name>;

The OUTPUT command identifies the task, associates with the task a single buffer which must be large enough to hold the complete record to be inserted and associates the relevant record schema with the task. Following the OUTPUT command a record can be built in the buffer by inserting fields using the ADD command, the user is not required to know the representation of the record in the buffer however the record fields will physically appear in the same order in which they were ADDed. After the record has been created in the buffer the WRITE command is used to transfer the record to the content-addressable store. Following the WRITE command further fields may be ADDed to the existing fields and further WRITE commands issued or the CLEAR command may be used to free the buffer space.

Example

An example is provided in 3.3.4.

DELETE Command

DELETE IF <selector> IN <record name>;
<selector>:= as defined for SEARCH command.

This command deletes from the content addressable store a single record occurrence for which the selector is "true". A status condition indicates whether a suitable record has been found and deleted. If there are a number of records which match the selector then the DELETE must be used repeatedly.

Example:

DELETE IF MANNO EQ 1726BM IN PERSONNEL;

CLEAR Command

CLEAR <task name>;

This command abolishes the named task freeing the buffers which were associated with it.

Example: CLEAR INRUN ;

C.4 Data Mapping Commands

The data mapping commands are used to define record schemas and to copy data between task buffers and "strings" or "variables" for computational purposes.

RECORD Command

RECORD <record name> ITEMS <LIST <<item name><attribute>>>
<attribute> := A/N

This command creates a new description of a record class (record schema). The item names are local to the record schema (i.e. the same item name can occur in two different record schemas and no relationship is assumed between the items). The exception to thus rule occurs for items defined by a special variant of the record command where record name takes the value "GLOBAL". The GLOBAL variant does not describe a record class, rather it identifies items which are global to the record schemas i.e. which have the same meaning for each record schema and are assumed part of all record schemas. An example of a global item might be "DAS address" if the CAS was being used in conjunction with a

Direct Access Store as indicated in 3.5.

The items in a record schema do not have any associated physical storage for data, they serve to identify particular fields within buffer storage allocated either to a SEARCH task or to an OUTPUT, WRITE task. Each item has an "attribute" indicating whether the corresponding buffer field may be copied into a "string" or a "variable" for computation purposes.

Example: See 3.3.2.

ADD Command

ADD < LIST <variable 1 string≫AS <item name> INTO <task name> ;

This command is used repeatedly to create a record in a buffer for output to the CAS (by a write task in the CAS Controller). The "task name" has previously been defined by an OUTPUT COMMAND which also specifies the buffer and the record name of the record to be output. The item names above must be local to this record name or a global. The ADD command adds the new fields to the record, the field "value" and "length" are derived from the variable or string. The sequence of fields in the record corresponds to the order in which they were ADDed.

COUNT Command

COUNT <item name> FOR <task name>{(<A.E>)}
INTO <variable>;

This command counts the number of occurrences of a field corresponding to the named item within the information returned by a task and present in the task buffers. There are two forms of the counting operation according to whether the <A.E> is present or not.

If the <A.E> is present the command counts the number of occurrences within the particular "result" (see SEARCH Command) indexed by the <A.E>. If there is no <A.E> the number of occurrences are counted across all the results.

Example: COUNT HOURS FOR PAYTASK (2) INTO TOTAL;
The number of occurrences of HOURS within the second result from PAYTASK is inserted into TOTAL.

COPY Command

COPY <item name> (<A.E>) FROM <task name>{(<A.E>)}

INTO <variable 1 string>;

This command is used after a COUNT command to select particular fields within the task results and copy their field values into variables or strings. There are two forms of the command corresponding to the two forms of the COUNT Command.

Example: COPY HOURS (TOTAL) FROM PAYTASK (2) INTO SUM ;

If this command had followed the example given under COUNT Command the effect would be to copy the last occurrence of the field hours, within the second result of PAYTASK, into the variable SUM.

C.5 Program Oriented Commands

PROG Command (Valid only in stream mode)

PROG <program name> FORMAL <LIST><formal names>>;
<program body>**

```
<replacement>        :=§<formal name>§
<valid segment>      := <CHS not §>|< replacement>
<valid string>       := <valid segment>|<valid string><valid segment>
<line>               := <valid string>;
<program body>       := <line>|<program body><line>
```

This command is used to store a program, it defines the program name and a list of formals. The command is terminated by two asterisks since semicolons occur in the program body. The parameters in the CALL command correspond to the formals defined in the PROG command and the appropriate parameter is inserted at each replacement before the line is passed to the interpreter as a command. No syntax check is performed on the program body as it is being stored.

```
Example:  PROG RUN1 FORMAL V1, V2, OP;
          LAB: TOT = §V1§ * §V2§;
          BILL, TOT = TOT §OP§ B1 ;
          IF (V1 - TOT) EQ GO LAB ;
          RETURN;
          **
```

DESCRIBE Command (valid only in stream mode)

DESCRIBE <record name>;

This command is used to interrogate the CAS Data Base description (which has been established by RECORD commands). The local items and attributes corresponding to the record name (global items – the record name "GLOBAL" is used) are displayed on the stream device.

DISPLAY Command (valid only in stream mode)

DISPLAY <parameter>;
<parameter>:= <variable>|<string>|<task name>

This command displays on the stream device the value of a variable or string in standard format or displays all the field values in a task buffer in tabular form with item names as column headings (see 3.3.4 Example 1 Part 3).

CALL Command (valid in stream or program mode)

CALL <program name> LIST <parameter>>;

<parameter>:= <CHS not ;,§>

This command calls the named program with the given parameters which replace the formals defined in the program. A program may be called with any number of parameters less that or equal to the number of formals.

Example: CALL PROGRA "JIM" + , OPEN (3) = 100, REP ;

RET Command (valid only in program mode)

RETURN;

This command returns control to the next command after the last CALL executed.

IF Command (valid only in program mode)

IF (<A.E>) <comp> GO <label>;

This command is a conditional jump according to the value of the arithmetic expression, the comparison operator indicates how the value is to be compared with zero.

Example: IF (TOTAL + 1) EQ GO LAB1 ;

If the value of TOTAL + 1 is equal to zero the jump is to be taken.

JUMP Command (valid only in program mode)

JUMP <label>;

This is an unconditional jump to the labelled command.

Example: JUMP LAB3 ;

MATCH Command (valid only in program mode)

MATCH (<S.E>), (<S.E>) GO <Label>;

This command is a condition jump according to the bit for bit equality of the two strings which must be of the same length. If the strings are equal the jump is taken.

Example: MATCH ("JOE"), (NAME(S)) GO LAB6 ;

Appendix D

Simulation Program Outline

The m-server queuing model (see 3.4.3) of the CAS was programmed in Fortran. A simple flowchart appears in Fig. D.1 through Fig. D.3 and the following description relates to these figures.

The simulation is "event-based" i.e. the variable CLOCK holding the simulated time moves in discrete steps corresponding to events. The total time for simulation is held in TOTAL (which was a parameter of the model) and CLOCK is initially zero. Two random variable generators were constructed (from standard FORTRAN random number generators) to provide the uniform service-time distribution and an exponential distribution of times between arrivals. (A poisson arrival frequency distribution means a negative exponential distribution between events). The mean values of the arrival and event distributions were parameters of the model. The m servers were represented by m variables SERVER one for each server (SERVER is an array but since each server is identical we do not bother to distinguish individual servers in the flowchart) m being a parameter of the model. A FIFO queue was programmed since the time spent in the queue was required. LENGTH and RESPONSE correspond to the length and response time histograms, WASTE accumulates the total idle time for all servers.

BLOCK A Fig.D.1 gives the overall logic, the model is driven by the two classes of event:- server events when a server has completed a service and an arrival event where a transaction arrives the (future) times at which the events occur are compared and the earlier event determines which loop will be taken. BLOCK B Fig.D.2 processes a server event. If there are no entries in the Q the server becomes "idle", otherwise the first queue entry is taken and the future server event time is placed in the corresponding SERVER variable. The LENGTH and RESPONSE histograms are updated. BLOCK C Fig. D.3 processes an arrival event. If a server is unavailable the arrival is queued otherwise an idle server is selected, his idle-time added to WASTE and a new future server event time inserted in the appropriate SERVER variable. The LENGTH and RESPONSE histograms are updated as indicated.

The simulation terminates when the CLOCK time passes the TOTAL time and the histograms LENGTH and RESPONSE are printed with computed mean values, WASTE is used to check that the server utilisation corresponds to that expected.

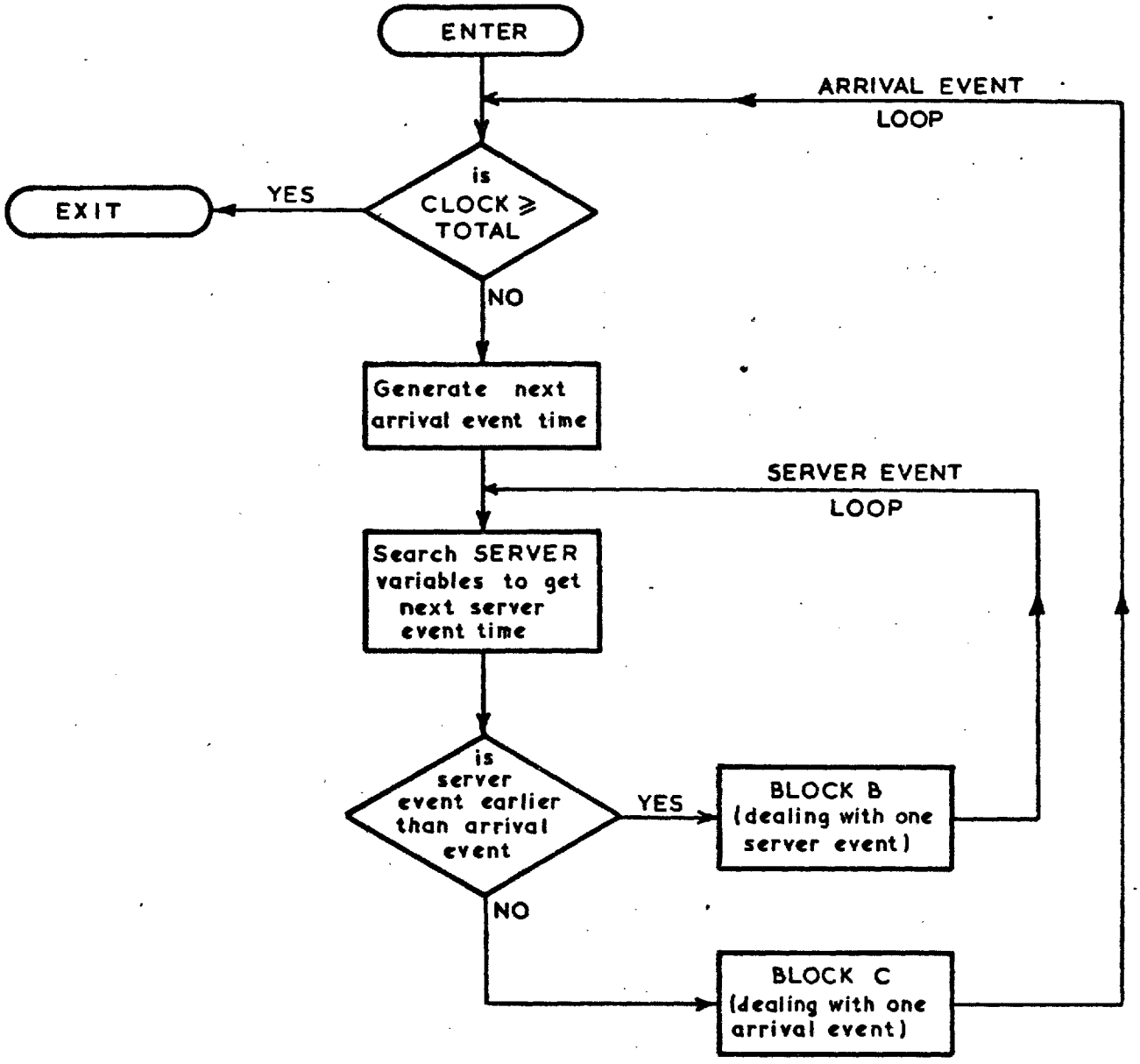


Fig. D.1 BLOCK A : Queuing Model

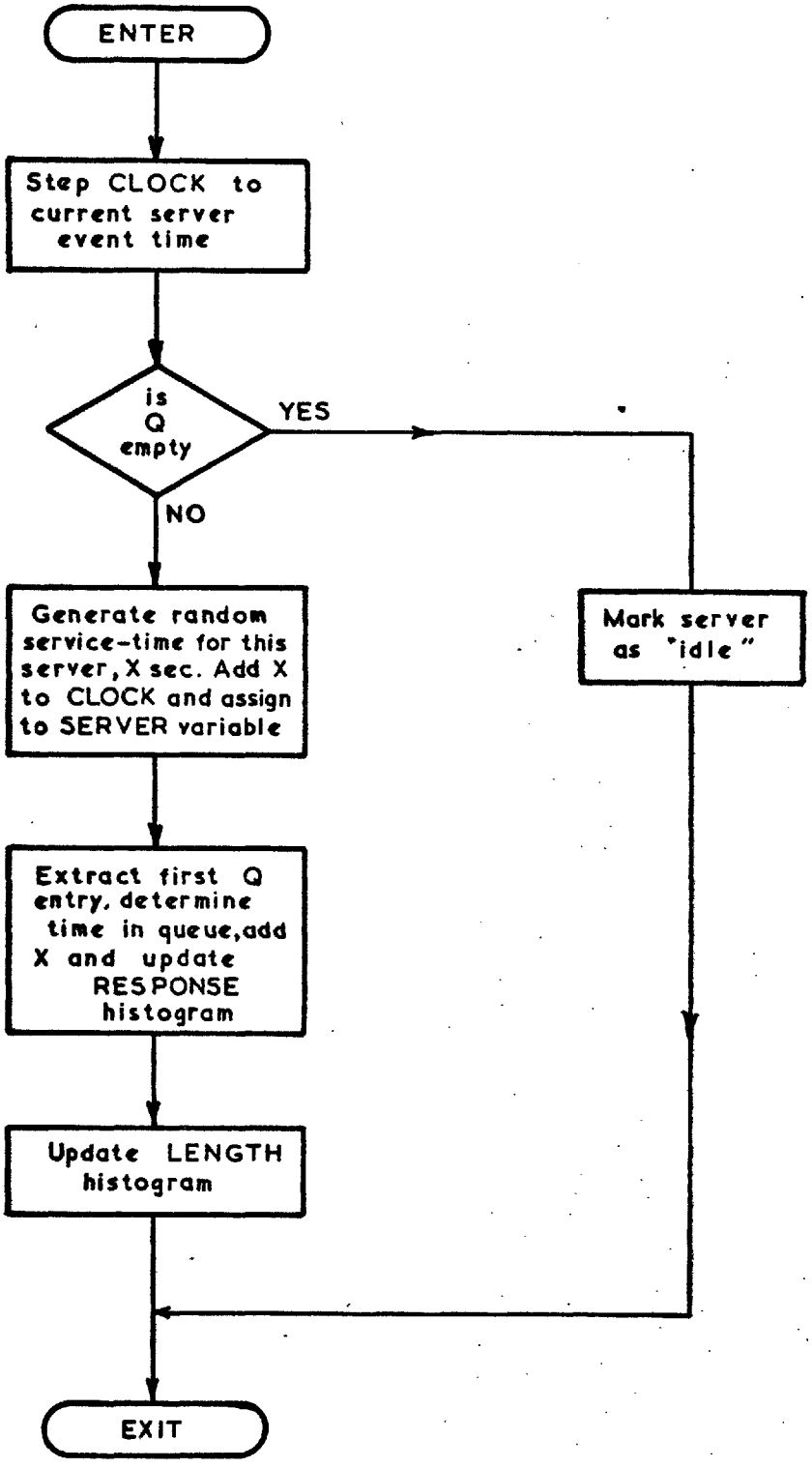


Fig. D.2 BLOCK B : Process One Server Event

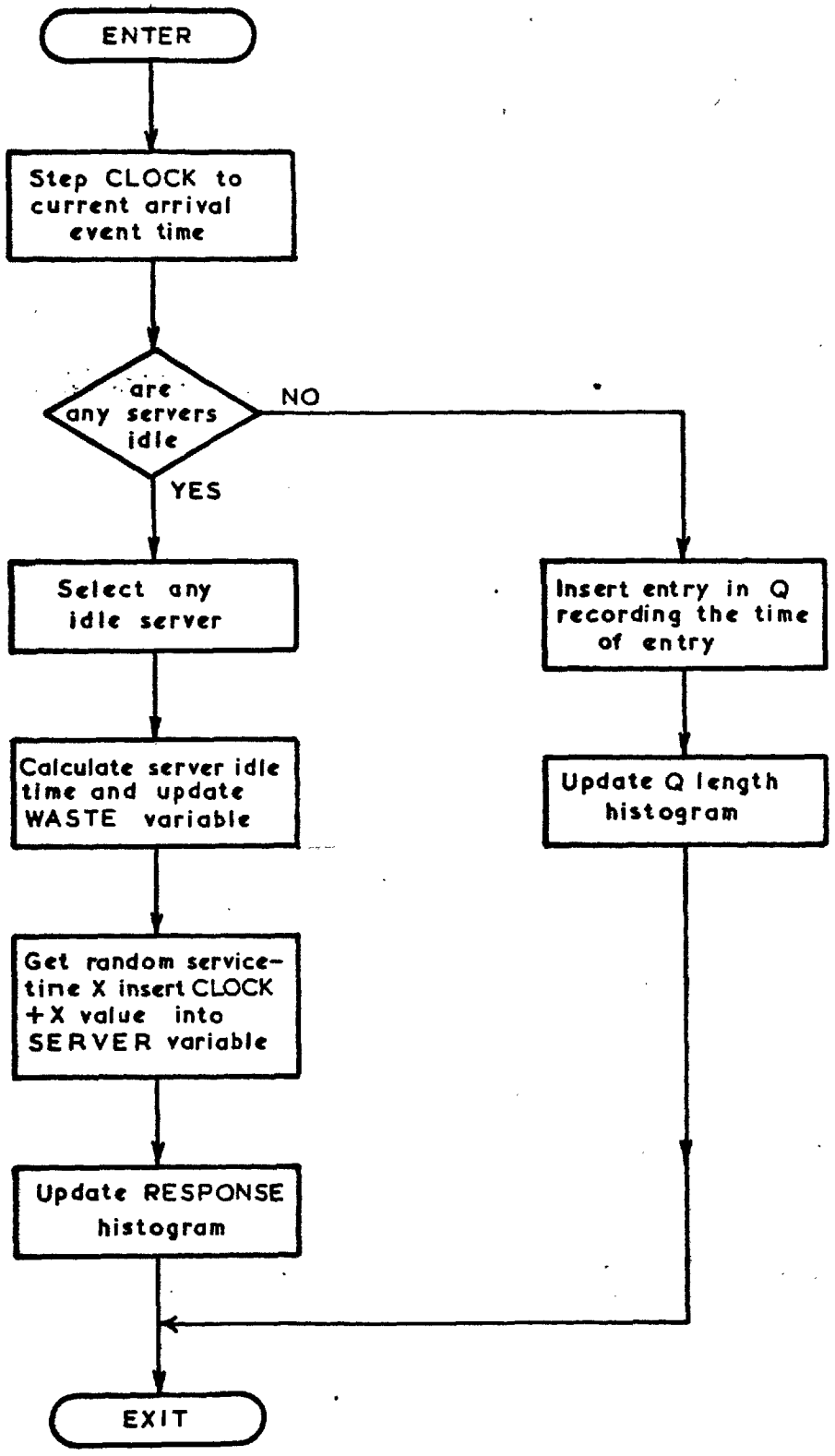


Fig. D.3 BLOCK C : Process One Arrival Event

Appendix E

Detailed CRH Language Description

E.1 Data Description

E.2 Write-File Schema Declaration

E.3 Read-File Schema Declaration

E.4 Update-File Schema Declarations

E.5 Core-File Schema Declaration

E.6 Record Schemas

E.7 Groups

E.8 Items

E.9 The Data Manipulation Language (DML)

E.10 Complete Names (including Complete Item Names)

E.11 OPEN

E.12 CLOSE

E.13 WRITE

E.14 READ

E.15 GMV

E.16 SEL

E.17 VSP

E.18 INIT

E.19 TYPE Declarations

E.20 Control bytes

E.21 Host Language Homonyms

E.22 Error Monitoring

E.1 The Data Description

The first lines in any symbolic program constitute the data description. The first line must contain the $FILE assembly directive to introduce the data description and define the size of the Data Area which is a vector of store within which file areas will be allocated by the CRH system. The Data Area must be large enough for the sum total of all file areas to be used. The $FILE directive has the following syntax:-

$FILE n (where n is an integer giving the size of the data space)

The data description is terminated by a line beginning with a single asterisk, the other characters on the line are unimportant (in this thesis a line of three asterisks is always used).

Between the $FILE directive and the terminator line appear a single list of DDL declarations, one to a line, describing all the objects required by the program.

Example:

```
$FILE  500
file schema - 1
file schema - 2
- - - - - - - -
file schema - n
***
```

The above Data Description causes a vector of 500 words of store to be allocated as a Data Area for file areas.

File Schema

Function:

To describe a sequential file, indicating how it is to be used by the program segment and describing the record classes of the file by record schemas.

General Form:

<File Schema Declaration>
record schema - 1
- - - - - - - - - - - -
record schema - n

Description:

There are four different sub classes of file schema corresponding to the four usages "write", "read", "update" and "core", all have the general form indicated above and are distinguished by different file schema declarations. The File schema declarations include user assigned identifiers or "names", these names must be unique within the data description. The list of one or more record schemas, describing record classes to be processed, is present in all classes of file schema but the interpretation of the record schemas can be different.

Identifiers occurring within File Schema Declarations serve two purposes they constitute "names" for the file schema but also serve to identify the corresponding physical file (except for core-file names). Each file name must have a homonym in the host language which has been associated with the appropriate physical file (See E.21). The four classes of file schema are described in E.2 to E.5.

E.2 Write-File Schema Declaration

Function:

To identify a file schema to be used by the program for creating a file.

Syntax:

$ <identifier> W

Description:

The identifier is used to name the file schema, the character "W" indicating a write-file schema. The declaration indicates that this file is to be created (i.e. records will be transferred only from the file area to the file using the WRITE function.

The list of record schemas forms a complete description of all the record classes on the file. Each record schema specifies the format of all records belonging to that class. The format is a simple derivation from the record schema and is illustrated in Fig. 4.12, Chapter 4. The number of file records within a class is not specified nor is the sequencing of records on the file.

The list of record schemas describing the formats of the records is stored in a "header" on the file but in a more compact coded from and is available to the CRH system when the file is subsequently read.

Example:

```
$FILA  W
record schema - 1
- - - - - - - - -
record schema - n
```

E.3 Read-File Schema Declaration

Function:

To identify a file schema to be used by the program for reading an existing file.

Syntax:

```
$ <identifier>  R
TYPE n
```

Description:

The identifier is used to name the file schema, the character "R" indicating a read-file schema. This is followed by a TYPE declaration indicating, by the value of "n", the number of record schemas in the list which follows. The TYPE declaration is an Implementation feature and apart from its declaration is of no significance, it will not be considered further. The read-file schema declaration indicates that an existing file (i.e. one which has been created using a write-file schema) is to be read, records will be transferred only to the file area from the file using the READ function.

The list of record schemas include descriptions only of those record classes required for processing. Furthermore, each record schema need only describe those parts of the records relevant to the current activity i.e. it may be a sub-schema (see 4.2.4) of the corresponding record schema used to write the file and stored in the file header. Correspondences between record schemas of the read file and record schemas used in writing the file are established by record schema name.

Example:

```
$FILB R
TYPE n
record schena - 1
record schema - n
```

E.4 Update-File Schema Declaration

Function:

To identify a file schema to be used in updating an existing file.

Syntax:

```
$ <identifier - 1> U
$ <identifier - 2> R
TYPE n
```

Description:

Identifier-1 is used to name the schema. Since an update operation on a sequential file involves reading an input file and writing an output file two identifiers are required. Identifier-1 corresponds to the output file and identifier-2 corresponds to the input file.

The TYPE declaration is used exactly as for read-file schemas and the list of record schemas in an update-file schema is interpreted in the same way as for read-files.

Examples:

```
$FILE K  U
$FILE J  R
TYPE
record schema - 1
record schema - n
```

E.5 Core-File Schema Declaration

Function:

To identify a particular file-schema to be used by the program simply for describing workspace. No input/output occurs with core files.

Syntax:

```
$ <identifier> C
```

Description:

The identifier is used to name the file schema, the character "C" indicating a core-file schema. This file schema name does not require a host language homonym which references the file, as is the case with all other file schemas. Therefore, a core-file schema is not associated with a physical file, it does however have a file area to be used as workspace for the program. The list of record schemas maps this file area exactly as for write-file schemas.

Example:

```
$FILQ  C
record schema - 1
- - - - - - - - -
record schema - n
```

E.6 Record Schemas

Function:

To describe a class of records.

General Form:

```
<identifier>
 member - 1
 member - 2
 - - - - -
 member - n
```

Description:

The identifier is the name of the record schema. This name must be unique within a file schema. The "members" must be either Group Declarations or Item Declarations (with certain restrictions described later), each record schema contains at least one member. The members form a tree-like structure where groups correspond to "nodes" and items "terminals" in the tree. This tree structure is illustrated in Figure 4.1 (Chapter 4).

Member - 1 of the record schema must be a two byte item to be used by the CRH system for the "control bytes". This is an implementation feature described in E.20 and will not be discussed further. Member -2 to (n-1) may be either groups or non-variable items (provided that they satisfy the rules for level numbers). Member - n must be an item, it may be a word, byte or a variable item (see E.8). The interpretation of a record schema by the CRH system depends on the class of file schema to which it belongs. Within write and core-file schemas the record schema is a complete description of a record class. For read and update-file schemas the corresponding record schemas may only be a partial description (sub-schema) of the record class (see 4.2.4).

Examples: See (4.2.4).

E.7 Groups

Function:

To identify a collection of items and, for repeating groups, to allow repetitions of the collection of items.

General Form:

<Group Declaration>
member - 1
- - - - -
member - n

Description:

The members of the group may be item or group declarations, so that tree like structures with groups as "nodes" and items as "terminals" can be declared. The members of each group in such a tree structure are indicated in Fig. 4.1. Each group must have at least one item as a member.

Simple Group Declaration

Function:

To identify a collection of items:

Format:

1 <identifier>

Description:

The identifier is the name of the group and must be unique within a record schema. "1" is the level number of the group.

The function of a simple group is to provide a method of referring to a collection of items using a single identifier (the group name). Simple groups can only occur in the DML as parameters to the GMV function (see E.15) which copies the fields corresponding to member items from one area of main store to another. An equivalent operation could be programmed directly, but less conveniently, by copying each item individually.

Repeating Group Declarations

Function:

To identify a collection of items occurrences of which are repeated a fixed number of times in physical records.

Format:

1 <identifier> (n)

Description:

The identifier is the name of the group and must be unique within a record schema. The level number is "1" and the number of repetitions of member items is given by "n"+1.

The repeating group performs the same function as the simple group in that it identifies a collection of items and can be used to copy fields in the same way with the GMV function. A more important aspect of the repeating group is that for each item owned by the group the corresponding record fields are repeated n + 1 times. However, the items may only be associated with one of the repeated fields at any particular time. Consequently there must be a DML function to select which repetition is to be associated with each item. The SEL function (see E.16) performs this selection taking, as parameters, the group name and an integer to index the required repetion. For items belonging to a nested collection of repeating groups the numbers of corresponding fields are given by the product of the repetitions for each owning group. Any particular field value can be selected by applying the SEL function to each of the owning repeating groups.

E.8 Items

To identify a particular data value or vector of data values in a record.

General Form:

l<identifier> <attributes>

Description:

The identifier is the item name and must be unique within a record schema. "l" is the level number. The attributes distinguish the three different sub-classes of item described later.

Items are always associated with a single vector of one or more data values at one time. The physical counterpart of an item is a record field. If an item is a member of a repeating group (see E.7, Repeating Group Declarations) there are many instances of the corresponding field in a single record, otherwise there is only one. In the case where there are many instances of a particular field the DML function "SEL" is used to select the particular instance to provide the data values for the item. References to items may be embedded directly in the host language using "complete item names" (see E.10) and allowing manipulation of the data values.

Numeric Data Item Declaration

Function:

To identify a fixed sized vector of one or more numeric data values.

Format:

l <identifier> Nn

Description:

"N" indicates a numeric data item where the corresponding data values are positive or negative integers. The number of integers in the vector is "n" + 1, omission of "n" indicates a single integer. The level number is "l".

Example:

```
2 VECT N2
```

this indicates a vector of three integers.

Character Data Item Declarations

Function:

To identify a fixed sized vector of one or more characters.

Format:

```
1  <identifier>  An
```

Description:

"A" indicates a character data item where the corresponding data values are single characters. The number of characters in the vector is "n" + 1, omission of "n" indicates a single integer. The level number is "1".

Example:

```
2 STRN  A4
```

this indicates a vector of five characters.

Variable Item Declaration

Function:

To identify a variable length vector of one or more characters and specify the upper bound on vector length.

Format:

```
1  <identifier>   Vn
```

Description:

The "V" indicates a variable data item comprising a variable length vector of characters. "n" gives the upper bound on the vector length (i.e. the maximum number of characters is "n" +1). The actual length of the vector can be adjusted using the DML function "VSP".

E.9 The Data Manipulation Language (DML)

The DML consists of a number of CRH Functions and a construct called a Complete Name (this includes complete item names).

CRH Functions have the general form:-

$ <function name> <parameter list>

There are eight different functions identified by name. The parameter list makes reference to a single CRH OBJECT (with the exception of the GMV function which can operate on two objects) to which the functions is applied. Each of the functions can only be applied to certain classes and sub classes of object as indicated in figure E.1 In addition to referencing CRH objects the parameter list sometimes contains other parameters, "values" and "switch vectors". Although the functions operate on individual objects they may have an effect on related objects. For example the READ function operates on a file schema but transfers a record into the file area altering the values of fields associated with items in the schema.

References to CRH objects from within the parameter list must be complete names (see E.8). For convenience, and to improve readability, if there is only one component in the complete name the $ character may be omitted. (Note: this is only permissable within the parameter list since the context indicates that a CRH name is required). For example the function "$WRITE $FILA" can be written "$WRITE FILA". CRH Functions can always be preceded by user-defined labels.

	File Schema				Record Schema	Groups		Field		
Commands	Write File	Read File	Update File	Core File	Record	Simple Group	Repeating Group	Numeric Field	Character Field	Variable Field
OPEN	VALID	VALID	VALID	VALID	-	-	-	-	-	-
CLOSE	VALID	VALID	VALID	VALID	-	-	-	-	-	-
READ	-	VALID	-	-	-	-	-	-	-	-
WRITE	-	-	-	-	VALID	-	-	-	-	-
GMV	-	-	-	-	-	VALID	VALID	VALID	VALID	VALID
SEL	-	-	-	-	-	-	VALID	-	-	-
VSP	-	-	-	-	-	-	-	-	-	VALID
INIT	-	-	-	-	VALID	-	-	-	-	-

Fig. E.1 CRH Function Usage

E.10 Complete Names (Including Complete Item Names)

Function:

To reference a single object of the program Data Description.

Syntax:

```
<component> :=   $  <identifier>      .
<complete name> := <component>|<complete name><component>
```

Description:

A complete name consists of one or more components each containing an identifier declared in the Data Description. The complete name evaluates to a single object and provides the mechanism to reference a particular occurrence of an identifier which may be duplicated in the Data Description.

The complete name is evaluated with respect to the Data Description by searching for each component in turn. The search for the first component begins with the first file schema name. When this has been located the search for the next component commences, and so on until the last component is located. This is the object referenced. Complete names may contain any number of components (bearing in mind the physical constraints such as maximum length of the input line). However, they need never contain more than three components (File name, record name and item or group name) to be able to reference any object unambiguously. Complete names can be classed according to the objects to which they evaluate e.g. a complete item name evaluates to an item.

The complete item name is a particularly important construct in the DML since it can be embedded directly in the host Basic Language as an operand of arithmetic and assignment statements allowing the host language to process data values contained in fields. No other class of name can be used in this way.

E.11 OPEN

Function:

To prepare the physical file and area for subsequent record transfer and record processing.

Format:

$OPEN <file schema name>

Description:

The OPEN function allocates a file area and positions the physical file device for the first record transfer, using the file homonym (see E.21) to reference the physical file. For core-files schemas only a file area will be allocated since no physical file is required. Prior to "opening" a file schema any reference executed to any object of that schema executed in CRH functions or complete names in the host language, would cause a program error. Directly after opening the file schema the objects owned by that schema may be referenced without error. The newly allocated file area will be set to zero and mapped into fields i.e. all items will be associated with zero fields.

Different types of mapping are set up for the different file schema types.

For write-file schemas the record schemas completely describe the record formats. The OPEN function writes a "file header" containing these record schemas. The file area is mapped into fields of each of the different record classes (see Chapter 4,Fig. 4.12, The Record Schema).

For read-file schemas an existing file is to be read. The read-file record schemas may be sub-schemas of the file header record schemas according to the rules of 4.2.4. If all sub-schemas are valid then the file area is allocated and mapping into the relevant fields is arranged (see Chapter 4, Fig. 4.12, A Sub-Schema).

For update-file schemas the OPEN function need only be applied to the first identifier of the update file declaration. The file header is copied

from the input to the output file and thereafter the input file is opened as for read-files.

For core files a file area is allocated and a mapping into fields similar to write-files is arranged. No file devices are referenced.

E.12 CLOSE

Function:

To terminate the use of the physical file.

Format:

$CLOSE <file schema name>

To improve readability a five character name is used although only the first four characters are significant.

Description:

The CLOSE function only affects the physical file, consequently if applied to a core-file it has no effect (no error will occur in this case).

For write-files an end of file marker will be written, the magnetic tape rewound and the device de-allocated (i.e. the file homonym no longer references physical file).

For read-files the CLOSE function may be applied at any time causing the magnetic tape to be rewound and the device de-allocated.

For up-date-files the output file is treated as a write-file and the input file as a read-file. The CLOSE function need only be applied to the first identifier of the update file declaration.

E.13 WRITE

To transfer the current record from the file area to the file.

Format:

$WRITE <record schema name>

Description:

The current record is transferred from the file area to the file. The record schema name identifies the class of record which is being transferred.

The command can only be applied to record schemas which belong to write or update-file schemas. This is checked during program assembly.

E.14 READ

Function:

To transfer a record from the file into the file area and to recognise the record class, or the end-of-file condition if no record can be transferred because of the end of file marker has been detected.

Format:

$READ <file name> , <switch vector>

Description:

The file name must belong either to a read-file schema or to the input file of an update-file schema. The switch vector is the identifier of a vector of program labels to which control may be transferred after executing the READ function. The first label corresponds to the end-of-file condition. The second and subsequent labels correspond one-one with the record schemas of the file schema, in sequence (Their values need not be the same as the record schema names).

After transferring a record into the file area the READ function indicates its record type or end of file condition as indicated in the example of

4.2.5, program 2.

E.15 GMV

Function:

To copy the fields associated with one collection of items into the fields associated with another collection of items.

Format:

<argument> := <item name> | <group name>

$GMV <argument - 1> , <argument - 2>

Description:

The field or fields corresponding to argument - 2 are copied into those corresponding to argument - 1.

The function treats word fields as byte fields (Each word corresponding to four bytes). The method of copying may be visualised in the following way:- All the fields corresponding to the items of argument - 2 are concentrated into a single byte vector (Packing bytes, occurring because all fields are word aligned, are not included in the vector). This vector is then copied byte-by-byte filling each field, corresponding to the items of argument - 1, in turn. Either the byte vector or destination fields may be exhausted at any time without causing an error.

E.16 SEL

Function:

To select a particular instance of those fields corresponding to items belonging to a repeating group.

Format:

$SEL <repeating group name> , <value>

Description:

The "value" may be of any of the Host Language constructs which evaluates to an integer e.g. a number, register name or codeword identifier corresponding to an integer.

The member items of a repeating group define a collection of fields which are repeated a fixed number of times. The SEL function allows a particular repetition of the fields to be selected, the "value" parameter indexing the required repetition. Bound checks are performed on the value at execution time.

Where repeating groups are nested the function can be applied independently to the individual groups.

E.17 VSP

Function:

To adjust the index of a variable length field.

Format:

$VSP <variable field name> , <value>

Description:

The value (which may be a number, register or host language codeword identifier referencing an integer) provides the new index of the variable field. Bound checks are performed at run-time.

E.18 INIT

Function:

To adjust the index of a variable item according to the size of the

variable field of the current record.

Format:

$INIT <record schema name>

Description:

This function is an implementation feature required because the CRH system does not automatically adjust the index of a variable item according to the size of the variable field of the current record. INIT may only be used on record schemas in updated read-files. After a record has been read and the record class recognised the INIT function should be used on the corresponding record schema if this contains a variable item.

E.19 TYPE Declarations

As the CRH system is currently implemented the user must include a "TYPE" declaration in every read or update-file schema (see E.3, E.4). The TYPE declaration should never be referenced by user in the DML, any attempt to do so will cause an assembly error. The declaration takes the following form:-

TYPE n

Where "n" is an integer giving the number of record schemas in the file schema. The TYPE declaration results in the allocation of storage for use by the READ function in recognising different record classes, its presence is needed purely for the implementation of the READ function and has no logical significance. The requirement for TYPE declarations is undesirable but can fairly easily be removed in any future implementation.

E.20. Control Bytes

As the CRH system is currently implemented the user must declare a two byte item directly after every record schema identifier (see E.6). This item will be used by the CRH system to hold control information. The user should never reference this item in the DML. If, however, the user assigns values to this item when using write or update files no failure of CRH system or user program will occur but the automatic

recognition of different record classes provided by the READ function may not operate consistently when the output file is subsequently read. The identifier for the control bytes should the same for all record schemas used in the computer installation. In all the record schema examples in Chapter 4 control bytes are declared by the following item:-

```
1 CON A1
```

E.21 Host Language Homonyms

The host language has facilities for declaring named objects belonging to different categories, for example "labels", "data vectors", "files" etc. The mechanisms for declaring these objects are not discussed in this thesis but the relationship of the names assigned within the host language to the names of CRH object is important. In general homonyms for any CRH objects can be freely declared and used in the host language parts of a program. No ambiguity arises because the "$" character always indicates the context i.e. whether a host language or CRH name is expected.

An exception to this rule occurs for file Schema names. The host language homonyms of these objects must correspond to file names. The CRH system expects the physical file corresponding to a file schema to be identified by its host language homonym. The host language has flexible facilities for assigning file names to physical files.

E.22 Error Monitoring

There are two classes of error detected by the CRH system assembly errors and run-time errors. These classes correspond to the standard basic language classes of error and are dealt with in the same way.

Assembly occurs line by line and generally (at the user's option) a line printer listing similar to that of Appendix F is produced. The first error detected on a line terminates syntax checking of that line and an error message is printed directly after the listing of the line. The code-generation part of assembly terminates, however all the subsequent program lines are read and syntax checked so that further errors are noted. A standard error message is printed together with the error number which distinguishes different sorts of error.

Run-Time errors occur during program execution and caise execution to be terminated. A monitor device can be nominated by the user to receive diagnostics during program execution. (Failure to allocate such a device causes the diagnostics to be written to a standard system monitor device). The monitor device may be any slow peripheral to be used as an output device for the program, when and if required the operating system will break-in to output the diagnostics. Run-time errors consist of an error message followed by a listing of the contents of the program registers. An example of a run-time error message is given in Fig. E.3.

Assembly Errors

There are five main types of assembly error:-

(a) Errors in individual object declarations in the lines of the DDL.

(b) Errors in the structuring of record schemas using level numbers.

(c) Omissions or invalid duplications of object declarations in declaring file schemas.

(d) DML syntax errors.

(e) Class of object referenced by parameters of DML inappropriate to context.

Run Time Errors

There are three main types of run time error:-

(a) Attempt to reference objects of a file schema which has not been opened.

(b) Failure to open a file because physical file unallocated, or (for update and read-files) invalid sub-schema found.

(c) Integer parameters out of range.

Appendix F

Listing of Program Example

```
0000 T1 SET #11
0000
0000 $FILE 70
$FILB W
RECA
 1 CON A1
 1 NAME A7
 1 NUMB N
 1 ROW (3)
  2 COL(2)
   4 ELMT A3
 1 SIZE N
 1 STR
  2 CODE A2
  2 VECT N2
RECC
 1 CON A1
 1 GRP(3)
  2 ATOT N
  2 STMP
   3 RG(2)
    4 NAME A3
   3 LOC A3
 1 VALS N1
***
0006 % THE ABOVE SCHEMA DESCRIBES A FILE WITH TWO RECORD CLASSES
0006 % "RECA" AND "RECB"
0006
0006 % OPEN THE FILE
0006 $OPEN FILB
0019
0019 % INITIALISE THE RECORD "COUNTERS"
0019 % X7 IS THE COUNTER FOR RECA,X6 IS THE COUNTER FOR RECB
0019 X6 LD 1
0020 X7 LD 1
0021 % INITIALISE X2 WHICH CONTROLS THE LOOP "MLP"
0021 X2 LD 3
0022
0022
0022 % THIS LOOP FIRST INITIALISES THE CURRENT RECORD AS A "RECA"
0022 % CLASS AND OUTPUTS THE RECORD,THEN INITIALISES A "RECC" RECORD
0022 % AND OUTPUTS.
0022 MLP:DUMP X2
0026 % HERE CALL SUBROUTINE "RCA" TO INITIALISE RECORD AS A "RECA" CLASS
0026 X1 J RCA
0028 X7 ADD 1
0030 % HERE OUTPUT RECORD AS A "RECA" CLASS
0030 LL21:$WRITE RECA
0046 % HERE INITIALISE RECORD AS A "RECC" CLASS BY CALLING "RCB"
0046 X1 J RCC
0048 X6 ADD 1
0050 % HERE WRITE RECORD AS A "RECC" CLASS
0050 LL22:$WRITE RECC
0066 UNDO X2
0067 X2 JNL MLP
0068
0068
0068 % HERE HAVE OUTPUT FOOR RECORDS OF EACH CLASS. NOW CLOSE FILE
0068 LL40:$CLOSE FILB
0083 STOP
0088
0088
0088 % HERE IS THE SUBROUTINE "RCA"
0088 % THE CURRENT RECORD IS INITIALISED AS A "RECA" CLASS USING THE
0088 % RECORD COUNTER VALUE OF X7
0088
0088 RCA:X4 LD SNAME!
0094 X3 LD "RECA NOO"!
0097 LL1:X4 MV X3
0101 X4 JL 2
0102 X3 JNL -2
0103 X4 ADD X7
0104
0104 X4 LD SNUMB!
0107 X3 LD 200
0109 X4 MV X3
0110 X4 ADD X7
0111
```

```
0111  X4 LD $VECT!
0114  X3 LD 0
0115  X3 ADD X7
0116  X4 MV X3
0117  X4 MOD 1
0118  X3 ADD 1
0120  X4 MV X3
0121  X4 MOD 1
0122  X3 ADD 1
0124  X4 MV X3
0125
0125  X4 LD $CODE!
0128  X3 LD "BAO"!
0131  LL2:X4 MV X3
0135  X4 JL 2
0136  X3 JNL -2
0137  X4 ADD X7
0138  X4 LD $SIZE!
0141  X4 MV 4
0143  X4 ADD X7
0144
0144  X4 LD 3
0145  X2 LD #25
0147  % HERE THE VALUES OF "ELMT" ARE INITIALISED, THE FUNCTION SEL
0147  % MUST BE USED ON BOTH "ROW" AND "COL"
0147  ROW:$SEL ROW,X4
0163  X3 LD 2
0164  COL:$SEL COL,X3
0179  X0 LD $ELMT!
0182  LL3:X0 MV X2
0186  X0 MOD 1
0187  X0 MV X7
0188  X0 MOD 1
0189  X0 MV X4
0190  X0 MOD 1
0191  X0 MV X3
0192
0192  X3 JNL COL
0193  X4 JNL ROW
0194   J X1
0195
0195
0195  % HERE INITIALISE CURRENT RECORD AS CLASS "RECC" USING X6
0195  % WHICH IS THE RECORD COUNTER
0195
0195  RCC:X4 LD $VALS!
0201  X3 LD 1
0202  X4 MV X3
0203  X4 ADD X6
0204  X4 MOD 1
0205  X3 ADD 1
0207  X4 MV X3
0208  X4 ADD X6
0209
0209  X0 LD 3
0210
0210  GRP:$SEL GRP,X0
0226  X4 LD $ATOT!
0229  X4 MV X0
0230
0230  X4 LD $LOC!
0233  X3 LD "LC"!
0236  LL4:X4 MV X3
0240  X4 JL 2
0241  X3 JNL -2
0242  X4 MV X6
0243  X4 MOD 1
0244  X4  MV X0
0245  X3 LD 2
0246
0246  NM:$SEL RG,X3
0261  % NOTE A COMPLETE NAME OF TWO COMPONENTS MUST BE USED TO REFERENC
0261  % THE "NAME" ITEM WITHIN THE GROUP "RG" BECAUSE THE ITEM "NAME"
0261  % ALSO OCCURS IN THE "RECA". SCHEMA
0261  X4 LD $RECCSNAME!
0264  X2 LD #2E
0266  X4 MV X2
0267  X4 MOD 1
0268  X4 MV X6
0269  X4 MOD 1
0270  X4 MV X0
0271  X4 MOD 1
0272  LL5:X4 MV X3
0276  X3 JNL NM
0277  X0 JNL GRP
0278  J X1
0279  END
```