ON IMAGINARY FIELDS, TOKEN TRANSFERS AND FLOATING CODES IN INTELLIGENT SECONDARY MEMORIES*

G. J. Lipovski
Department of Electrical Engineering
The University of Texas
Austin, Texas

ABSTRACT

In analyzing two implemented intelligent secondary memories, CASSM and RAP, we recognize a common mechanism which is here called an imaginary field. The mechanism can be generalized to suggest further design possibilities. It further explains the complex and controversial CASSM mechanism, "pointer transfer" in terms of separate but related principles---imaginary fields, token transfers, and floating codes---such that further designs can utilize some or all of the techniques.

1) INTRODUCTION

The need for special processors to manage large data bases---non-numeric processors---is well documented already [1]. Two fundamental approaches to designing these processors are the "evolutionary" and "revolutionary" approaches, to use Tomás Lang's terminology. The "evolutionary" approach attempts to replace a part of a few parts of the software by hardware. Typically, a hardware sorting device can considerably improve the performance of a data base management system. While we are generally optimistic about the successful near-term application of the "evolutionary" approaches, we wonder that it is like an inner tube with patches on patches; a fundamental change will eventually be necessary. Hence we study the "revolutionary" approach in earnest.

Two fundamental assumptions are being made. For hardware reasons we opt to search the data where it is, effectively making the secondary memory our theater of activity. For software reasons, we develop our instruction set on the requirements of one or more data base models. Two systems, CASSM [2, 3] and RAP [4] have been implemented. This paper analyzes some techniques which contribute to, and indeed may be necessary for, the efficient use of such machines.

At the outset, the classical definitions of architecture, organization and realization will be offered to clarify issues addressed later. The architecture of a machine is essentially the programmer's view of a machine. The organization is the block diagram and the realization is the logic family (e.g. TTL, CCD memory, etc.) and board layout, etc. If a concept need not be explained to a programmer so that he can effectively program the machine, it is not architectural. We are now primarily interested in architectural concepts, such as developing the instruction set on data base models, and on organizational concepts, such as developing the system as an intelligent secondary memory.

2) ORGANIZATION AND STORAGE STRUCTURE

In order to avoid passing large amounts of data through a data channel between primary and secondary memory, we opt to search data directly in secondary memory. In the interests of utilizing large scale integrated circuitry, we put a small but identical processor, a "microprocessor" specialized for data base management functions, on each head of a fixed head disc. In a different realization, clearly, the disc track can be a CCD or magnetic bubble memory. While we would like to fit the entire data base

* This paper was supported in part by NSF grant GJ 43225.

within this intelligent memory, in the foreseeable future we can at best hope to page in a reasonably large part of the data. The remainder will have to be in a tertiary memory. Note that this does not necessarily imply a large input/output channel between secondary and tertiary memory. In a realization employing an IBM 3330 disc, a feasible tertiary memory, the heads on all surfaces are moved together so that a cylinder is under them at any given time. That cylinder can be effectively an intelligent secondary memory.

The data on each disc track is searched sequentially even as all tracks are simultaneously searched. The same search or update operation is executed on all tracks, this being classified as an SIMD (single instruction stream, multiple data stream) organization according to Flynn's classical terminology [5]. The data on the track has been called a segment and this general approach where each segment is searched sequentially has been called segment sequential.

The storage structure for a segment-sequential memory like CASSM is defined in terms of real fields, stored in the memory, and imaginary fields, generated by hardware. Herein, we define a data base as a collection of files. See figure 1. A file is part of the data base that is searched exhaustively, for example, by putting it wholly in a CASSM or RAP secondary memory. A file is a sequence of fixed length words which is partitioned into variable length contiguous subsequences called records. For relational data bases, a record is a tuple. In RAP, segments are architectural concepts because the programmer has to be aware that only tuples belonging to the same relation can be stored in a segment; however, he can take advantage of this fact to search only the required tuples. In CASSM, segments are transparent, architecturally, since the programmer does not know which segment this data is on even as records overlap segment boundaries and concurrent hardware garbage collection moves data across segment boundaries. This may make it difficult to conduct several independent searches on several parts of the data base. However, the task space concept described in [6] can be used to advantage so that within a task space, segment boundaries are transparent, but task spaces are so created that several independent searches on several parts of the data base can indeed by executed concurrently.

3) IMAGINARY FIELDS

The effective content-addressable word in this storage structure is actually only partly stored, to increase storage efficiency. The fixed length word that is stored on the disc is partitioned into real fields, and has associated

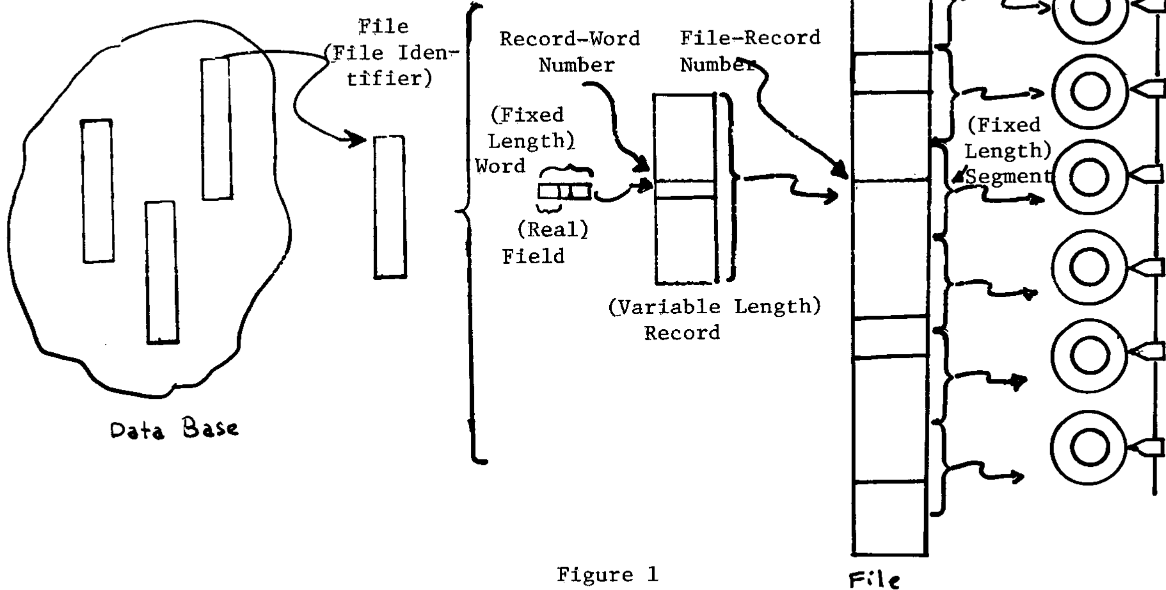

Figure 1
Data Base Organization

with it hardware-generated imaginary fields. Both real fields and imaginary fields can be content-addressed, but only real fields are stored in the memory. Some imaginary fields for a word w are: 1) the file identifier, which names the file containing w in the data base, 2) the file-record number which equals the number of records above the record containing w in the file, 3) the segment-record number which equals the number of records above the record containing w on the segment containing w, 4) the record-word number which equals the number of words above w in the record containing w, and 5) the segment-word number which equals the number of words above w in the segment containing w. Some of these imaginary fields are architecturally significant since the programmer is aware of them, while others are only organizationally significant.

Imaginary fields have appeared in CASSM and RAP. In CASSM, the segment-record number is maintained by a counter that is cleared at the beginning of each disc revolution and is incremented each time a new record is met. The file-record number is derived from the segment-record number by adding the segment-record number to a base address, where the base address of a segment is the sum of the segment-record numbers of all segments above it at the end of a disc revolution. The segment-record number is an organizational concept while the file-record number is architectural. Simple hardware to maintain these imaginary fields is described in [7]. In RAP, record-word numbers are maintained by a counter that is cleared at the beginning of a record and is incremented when each word is processed.

One of the problems that we stumbled around within CASSM extends simply to distributed systems. Whereas RAP outputs whole records, CASSM outputs only selected words of a record. We were unable, however, to find a really efficient output technique because it was difficult to keep words from a record together and in order as they are output. Clearly, however, if we included imaginary fields such as file identifier, file-record number and record-word number in the output word, the host processor can effectively sort out the (expectedly) short bursts of output words. We are only beginning to understand the significance of this notion in distributed systems, but here is our initial analysis of the problem. Extending this to distributed systems, communication between different CASSM processors has to be in terms of all architecturally significant imaginary fields. Consequently, the problem in maintaining efficient transfer is actually one of suppressing wherever possible the imaginary fields from architectural to organizational concepts to avoid having to transport them through the network.

We are also sensitive to the effect on support software. Because architecturally significant imaginary fields are visible to the programmer, the support software must maintain coordination between them. Specifically, insertions or deletions generally require that the query language compiler track such imaginary fields as segment-word number.

Clearly there is a tradeoff: Imaginary fields increase storage efficiency because they don't have to be stored. However, they restrict freedom because they are generated according to a fixed algorithm embedded in hardware. The trick is to find more clever algorithms than simply counting words to recover the lost freedom while gaining storage efficiency. For instance, specially marked words could load the counters maintaining an imaginary word so as to permit the count to skip over unused word positions or reset to store multiply defined postitions. Also, there is a need, we think, to build higher level processing functions into hardware to reduce imaginary fields to organizational significance to avoid having to output them.

4) TOKEN TRANSFERS

Imaginary fields also become very useful when combined with token transfers, which are now discussed. A token transfer uses the output mechanism of an intelligent secondary memory to collect values of real or imaginary fields of selected words, and a one bit wide random access memory (RAM) to record the tokens. A setting transfer uses the value as a RAM address to set the addressed bit, a read transfer uses the value as a RAM address to read the bit, and a read-clear

transfer is a read transfer that also clears the bit. The bit read from a read transfer is used to mark the word whose field provided the address for the read. A setting transfer followed by read transfers in which the last one is a read-clear transfer is a token transfer. The RAM has to be only large enough so that each value corresponds to a valid RAM address. If values (code words) are assigned carefully, the RAM can be kept small.

Consider this example, as shown in figure 2. For simplicity, we consider only one real field per word and we aim to mark the words in set B that have the same values as those in set A. The values of code words in the real field for each word are shown in figure 2. These marked words might later be output or rewritten, depending on future instructions. The words in set A are read out the output channel as though to be sent to the host CPU. Instead, they are sent to the RAM to set bits (setting transfer). Bits 2, 5, 7 and 9 are thus set. This is done as fast as output can normally be done---perhaps in one disc revolution. In the next revolution, the words in set B are read out, and are marked according to the bits in the RAM as they provide addresses for a read-clear transfer.

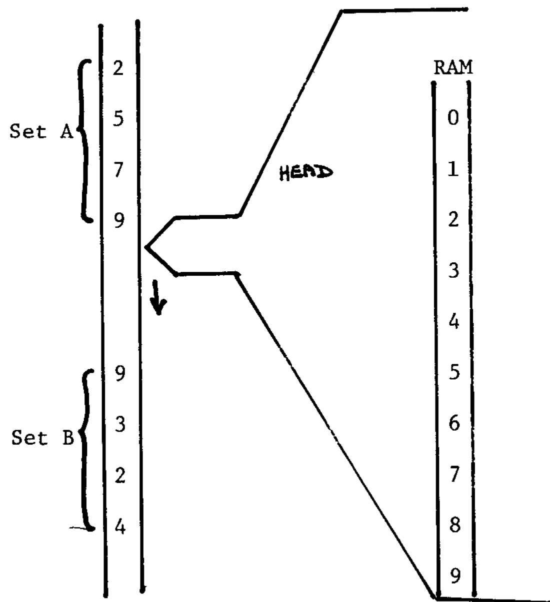

Figure 2
Token Transfers

The word having code word 9 will be marked since it reads a 1 from the RAM. The word having code word 3 is not marked since it receives a 0 from the RAM. The word having 2 is marked and the word having 4 is not marked. This read-clear transfer again is as fast as the output channel, and can be done in one disc revolution. In just two revolutions, the set intersection is marked in the set B.

RAP can use a setting transfer from real fields, and a read-transfer from real fields to effect their implicit join as fast as words can be read out from RAP. This overcomes one of the most serious flaws with the current RAP design. CASSM uses token transfers from and to the segment-record number imaginary fields to collect the results of a search in a record into the first word of the record where it is generally stored.

It should be observed that token transfers depend upon using code words as opposed to character strings for fields that may transfer tokens, and these code words should be carefully assigned and maintained to keep the size of the RAM small. This issue is taken up again later in this paper. However, it lends considerable power to functions that transfer tokens between records, in the relational, hierarchical, and especially the network data base models.

5) FLOATING CODES

Finally, a floating code is a real field that has a value defined by an imaginary field, where insertions or deletions may change the imaginary field values which require corresponding changes in the real fields. Consider a floating codeword associated with the file record number. If record i is deleted, for instance, all floating code words greater than i must be decremented. This can be accomplished by hardware in just one revolution of an intelligent secondary memory. Note that if there are n distinct codewords, hardware can automatically assigns number 0 to n-1 for these codes.

Consider the following example, which is used in CASSM. Suppose 'ALPHA', 'BETA' and 'GAMMA' are three items in a file in a data base. The character string 'ALPHA' is stored in the topmost record of the file, with file-record

number 0, 'BETA' and 'GAMMA' are stored in the next two records, with record numbers 1, and 2. The record numbers, as floating code words, are generated by hardware. Wherever the string 'ALPHA' might appear elsewhere in the file, the floating codeword 0 is used. Note that this improves storage density especially if the character string is long. Moreover, it permits the use of efficient token transfers between items 'ALPHA' in different records. The codes word floats because insertion/detection is easily done, as follows. Suppose 'BETA' is deleted. Then as the record containing the character string 'BETA' is deleted, all floating codes having value greater than 1, the codeword value of 'BETA', are simultaneously decremented. 'GAMMA' will now have codeword 1, and so on. Insertion can also be accomplished in reverse order in CASSM. The hardware can effectively maintain compact code words for floating codes.

Input and output is also effectively handled. Consider a query where the operand 'ALPHA' is a character string. A fast hardware algorithm described in [7] first marks all characters 'A' in character strings. This takes one revolution. Then, for each mark, the mark is erased and the subsequent characters of the comparand 'LPHA' are compared with the characters in the record. This second step may take more than one revolution if a check of the comparand is in progress when another mark is encountered, that mark being handled in a later revolution. When a perfect match is detected, the file-record number of the record containing the string is obtained. This implements the common string search subroutine effectively in hardware. Output is easily done. The floating codeword is transferred, via token transfer, to the record, and the string is output. This output mechanism is slightly slower than outputting character strings, but uses only hardware already valuable for searching, namely the token transfer hardware.

Floating code words are one answer to the question: Shall the machine process raw character string text or more efficient code words. Its answer is yes---to both questions. The advantages of both formats can be obtained.

6) CONCLUSIONS

Although imaginary fields have been used in earlier designs, the recognition that these earlier "tricks" are instances of the same fundamental technique lends understanding to these designs and should lead to better designs in the future. The isolation of token transfers from imaginary fields enables them to be applied to real fields to improve the performance of RAP. The further isolation of floating codes identifies a mechanism which can be used with imaginary fields, but can be omitted if found undesirable. These fundamental mechanisms break down the controversial "pointer transfer" of CASSM to clarify its behavior.

7) ACKNOWLEDGEMENTS

Most of these ideas were generated while CASSM was being designed. I am deeply indebted to my former colleague, Stanley Su, and to two especially bright graduate students, George Copeland and Ahmed Eman whose ideas, criticisms and questions extensively contributed to the design of that advanced machine. I am no less indebted to Stew Schuster, with whom stimulating interchange has lead to better understandings of both CASSM and RAP.

REFERENCES

1. Lipovski, G. J. and Su, S. Y. W., "On Non-Numeric Architecture," Computer Architecture News, Vol. 4, No. 1, pp. 14-29, March 1975. (Reprinted in: SIGIR FORUM, Vol. X, No. 1 (Summer 1975) pp. 5-20).

2. Healy, L.D., Lipovski, G.J., and Doty, K.L., "The Architecture of a Context Addressed Segment Sequential Storage," Proc. FJCC, 1972, pp. 691-701.

3. Su, S.Y.W., Lipovski, G.J., "CASSM: A Cellular System for Large Data Bases," Proc. Intl. Conf. on Very Large Data Bases, Farmington, MA, Sept. 1975, pp. 456-472.

4. Ozkarahan, E.A., Schuster, S.A., Smith, K.C., "RAP--An Associative Processor for Data Base Management," Proc. NCC, AFIPS, Vol. 44, May 1975, pp. 379-387.

5. Flynn, M.J., "Some Computer Organizations and Their Effectiveness," IEEETC, Vol. C-21, No. 9, Sept. 1972, pp. 948-960.

6. Lipovski, G.J., "On a Varistructured Array of Microprocessors," IEEETC, Vol. C-26, No. 2, February 1977, pp. 125-137.

7. Bush, J.A., Lipovski, G.J., Su, S.Y.W., Watson, J.K., and Ackerman, S.J., "Some Implementations of Segment-Sequential Functions," Proc. 3rd Ann. Symp. Comp. Arch., Jan. 1976, pp. 178-185.