KEY ADDRESSING OF RANDOM ACCESS MEMORIES
BY RADIX TRANSFORMATION

Andrew D. Lin
Information Storage Systems
IBM General Products Division
Development Laboratory
San Jose, California

INTRODUCTION

The addressing procedure for nonsequentially stored data in large capacity random access memories is the automatic allocation of storage position to each member of a data set. Each member, termed here as a record, is identified by a unique description or key which is part of the record. Key addressing then is the execution of a mapping function which mathematically relates the key of a record to its location in storage. To store or to retrieve a record, its key is presented to the addressing facility which then generates an address to which the memory will then access.

Various addressing techniques have been used in the past and are being used at present. Typically, however, they have been individually tailored to the application. The growing usage of large capacity random access memories is placing practical urgency on a unified approach to the addressing procedure. A unified approach implies freedom from the parameters, to be discussed later, which have governed the design of past techniques. A further consideration would recognize that random access bulk storages will increasingly appear as the central machine element in information storage systems in contrast to its peripheral role in present data processing systems. This then calls for a unified solution which lends itself simply and economically to hardware implementation yet permits its execution by stored program. The subject investigation was oriented toward that objective.

The Storage Address Set

The storage devices within the purview of this paper are large capacity random access bulk memories. In general, there is no difference in the addressing problem of a memory which depends on mechanical means for accessing and of a memory which depends on electronic means for accessing. In practice, however, mechanical access time is several orders of magnitude greater than electronic access time; hence, it becomes more significant in the former to reduce the number of seek cycles of the actuator for the average record in the data set.

These memories are usually organized in a binary or decimal hierarchy to agree with the radix of the arithmetic unit of the processor of whose system the memory is destined to be a member. The storage is generally divided into $q^n$ subspaces where $q$ is the number base of the organization structure and $n$ is some integer. The latter, of course, corresponds to the number of positions in the address register.

A subspace or group of subspaces is termed a bucket if, once the access mechanism has arrived, its entire contents may be scanned without further change in the contents of the
address register. For mechanically accessed memories, buckets tend to contain a plurality of records. A good reason for this is that the development of higher bit density in magnetic recording is advancing more rapidly than that of improved resolution and speed of access mechanisms. Hence, the tendency is for bucket capacities to increase in the future since a larger share of the burden of search will have to be shifted to the faster scan-time capability within the bucket and away from the slower access mechanism. This shift is necessary to strike a reasonable balance between access time and scan time.

Random access memories of the current generation are location-addressed rather than content-addressed. A bridging function is therefore necessary to associate the keys of a data set with memory locations. If, in the rare case, the key happens to be directly usable as an address or if the predetermined address is appended to the key, then “direct addressing” is possible. If the association is by means of arbitrary assignment of locations to keys, the process is called “table look-up”. Table look-up may consist of one or more stages with the stages stored either in internal memory or in the bulk storage or a combination of both. In “key addressing”, the key is operated on by a mathematical routine or algorithm to generate an address. In effect, then, key addressing makes the random access memory appear as if it were content-addressed, based on a record’s unique descriptor. A machine address set has these stable characteristics: It is numeric, a solid integer set in its range; consecutively ordered, and does not change with time. These are in sharp contrast to those of a data key set, which will be discussed next.

The Data Key Set

The data key set in typical random access memory applications has the following characteristics. The length of keys, of course, varies from data set to data set, with the maximum ranging up to the neighborhood of 20 characters. Members of a given key set frequently differ in length, with blanks often appearing in various positions of the allotted field. Character positions may be differently restricted as to allowable symbols such as numeric, alphabetic, alphanemic and special signs; i.e., keys may very well be mixed radix representations.

A data key set is typically a small fraction of the total allowable combinations or parent set which the key length and the source alphabet permit. If the parent set were arranged in collating sequence, it may be seen that the keys, as a subset, generally populate the range in an irregular manner, giving the impression of uneven clusters separated by huge gaps. In part numbers, for example, this is due to the classification logic adopted by the user; in English names, on the other hand, the phonic requirements and the relative incidence of vowels and consonants contribute to clustering in the spectrum of all possible combinations.

To a user, then, which specific subset in the parent set presents itself as the key set is probabilistic. Furthermore, in a dynamic application, the cluster and gap configuration will change with time because of accretions and deletions in the data set. In view of the above, probabilistic techniques must be used to provide the association or storage mapping function of keys to addresses.

Occupancy Distribution Considerations

In key addressing, the mapping procedure must distribute the keys of a set over the memory bucket addresses as evenly as possible. Ideally, the user desires, if the need arises, to load his memory to 100% of capacity without special fitting operations requiring complex programming; he also wants to find a record in exactly one access cycle. In other words, he would like to find his record always at the address which the mapping algorithm generated, i.e., in its home bucket. Both requirements imply a perfectly even distribution, which a probabilistic technique on a nondeterministic key set can only approach. The proper performance objective, therefore, is to achieve a mapping that is equivalent to what a randomly distributed set of elements would experience if mapped into a set of buckets, each having equal probability of being occupied. The expectation then would be a Poisson distribution. Clearly, the addressing algorithm sought should provide the conditions which would permit a similar expectancy. In short, the criterion of mapping performance should be the Poisson distribution.
**Bucket Overflow Considerations**

A key addressing technique involves two stages: first, a mapping procedure to allocate the records as evenly as possible among the buckets; second, an overflow procedure to divert the overflow into trailer buckets. Once the occupancy distribution is made, the bucket capacity, \( C \), determines the extent of the overflows. Obviously, the poorer the mapping performance, the greater the burden left for the overflow routine. The effectiveness of the latter lies in its added levelling effect on the distribution. This implies, for the average record in the data set, a reduced seek factor, \( S \); i.e., the number of seek cycles of the access mechanism to trace it to its overflow location. The criterion of merit for the composite procedure of mapping and overflow disposition is most usefully expressed as the percentage, \( R \), of the data set that is lodged into home buckets. The complement of \( R \), of course, is the overflow percentage, \( F \), that finds placement in trailer buckets. It might be added here that in applications where a high percentage of activity is confined to a small portion of the data set, the overflow problem becomes almost insignificant when the most active records are loaded into their home buckets and the less active ones are loaded into trailer buckets.

**Parameter Relationships**

The qualitative relationship among the parameters in an addressing technique are as follows: Consider any data set of \( N \) records occupying a fraction \( L \) (load factor) of the total space in memory of \( K \) buckets, each having a capacity of \( C \) records. For any memory of given \( K \) and \( C \), the seek factor improves (i.e., decreases) with a drop in the load factor. This may be seen in a preview of Figure 5. For a memory having product \( KC \) constant, but with \( K \) and \( C \) inversely variable, the seek factor improves with higher bucket capacity. A glance at Figure 4 also indicates the relationship between the load factor \( L \), the ratio of data records to buckets \( N/K = b \), and \( R \), the percent of the data set lodged into home buckets. For a given \( L \), \( R \) improves with a higher value of \( b \). These are relations to be borne in mind in subsequent pages of this paper.

**Source Alphabet Considerations**

A key is converted from its human-readable form to its coded representation before it can be machine-processed by an address transformation technique. This processing is arithmetic and, therefore, must deal with numeric digits. With nonnumeric characters in a key, the user faces the alternative of recognizing only the numeric component of the coded representation (for example, in the Binary-Coded Decimal code, stripping the two zone bits from the 6-bit configuration) or of devising some method of having all bits participate in the calculation process. If, as has been done frequently in past techniques, the nonnumeric bits are ignored, then the effect is to impose on the human-readable key set a condition where certain bit-coded subsequences have a one-to-many correspondence with human language characters in the key. This, of course, intensifies the clustering effect already present in the human readable key set. Clearly, this should be avoided in any addressing algorithm.

**System Considerations**

In planning an application for a random access bulk storage, the user has a certain latitude in assigning values to the critical parameters. If mapping performance were predictable in relation to these parameters, planning would become more systematic. For example, a user having chosen a certain load factor, \( L \), and a certain ratio of records to buckets, \( b \), could with confidence estimate the approximate percentage of the data set that would overflow. The mapping algorithm should, therefore, be reasonably consistent and predictable in performance for a given set of parameters.

Again, from the user's viewpoint, it is highly desirable that an addressing technique be directly applicable to any key set as found. In other words, the user would be glad to be freed of the task of preanalyzing and experimenting with the structure of a new key set to discover the series of procedures which will "randomize" it. He would prefer a general-purpose scheme which he can immediately apply to his key set with the high expectation of getting a transformed, randomly distributed set.

The primary interest in this paper on hardware embodiment of the addressing algorithm is based on the following system considerations. As a self-contained addressing facility, it can be integrated into a variety of system environments, both processor-centered and storage-centered. In the former, it would serve to over-
lap the addressing procedure with the main processing program. Additionally, it may serve as a time-shared device placed between one or more processors and an array of bulk storages (having same or different parameters). If additionally the hardware embodiment of the addressing algorithm should permit transformation of the key optionally into binary or decimal addresses, a further system advantage is gained. This would relieve bulk storages of the design restriction that its address radix be necessarily the same as the computing radix of the processor.

Storage-centered systems will, in general, not be equipped with as high a level of arithmetic and logical capability as processor-centered systems. A separate addressing facility would make it possible for them to become truly autonomous systems, not necessitating the association of a general-purpose processor to enable use of bulk storage for strictly retrieval purposes.

Summary of Requirements

To the above considerations, others may be added to compile a list of functional objectives. These are proposed here as the basis for any unified solution to the addressing problem.

1. The addressing algorithm should distribute the records among the memory spaces as evenly as is probabilistically possible, with the Poisson distribution as the objective.
2. The composite procedure of mapping and overflow tracing to store or retrieve the average record should be fast in terms of a low number of seek cycles.
3. The algorithm should be universal in application for all keysets. It should be independent of source alphabet, machine character code and key length.
4. The algorithm should have the element of predictability in the sense that a prospective user, electing specific values for some of the principal parameters (load factor and keys-to-buckets ratio) can have high expectation of a certain mapping performance, i.e., the percentage of the data set that will overflow.
5. The algorithm should be directly applicable to any key set, as found; i.e., without pre-analysis or pre-editing.
6. The algorithm should be implementable as independent hardware simply and economically, preferably using a systematic sequence of elementary operations. Time of execution should be a relatively low multiple of the data transfer rate of the using system.
7. Hardware embodiment of the algorithm should permit the option of decimal or binary output addresses.

Proposed Solution

The solution offered here for key addressing is briefly as follows. The key is introduced to the mapping process as a binary sequence, is interpreted as 4-bit digits and radix-transformed into a magnitude expressed in address radix $q$, where $p$ is relatively prime to $q$. Truncation yields the memory address. The rationale is given below.

As a first step in meeting the solution requirements, a common basis of representing all keys must be provided so that they may appear as a standardized input to the address-generating process. It has already been noted that human and machine language formats contribute to structuring in the key set. Eliminating the character partitions and expressing the coded form of a key as a binary sequence, i.e., a continuous train of ones and zeros, would reflect this compound structuring. The key representation is now an amorphous binary array without the attribute of magnitude. It awaits a choice of radix and of grouping to quantify it. The rules of quantification and the subsequent processing should transform the original key set into a randomly distributed set of entities. They should also permit a systematic iteration of elementary operations to permit economical hardware implementation as well as stored program execution.

A grouping of 4 bits was chosen as the unit of processing; i.e., as the byte size. This is reasonable since addresses are generally numeric and hence require 4 bits for a decimal digit. At the same time 4 bits is compatible with a binary address radix, since a magnitude expressed in a radix that is an integral power of 2 can always be read out as a pure binary number.

The results of processing successive bytes should be cumulative in the hardware implementation. Compression to the range of the addresses should be progressively done, pref-
erably by the simple operation of truncation by the physical bounds of the device itself.

To evolve our transformation algorithm, we start with the hypothesis that the principal phenomenon of key sets is the tendency to cluster in various ways and that this is the only significant reason which causes key sets to deviate from randomness.

We observe that a sequence of keys may occur for which all but a few bits of its coded representations remain constant. Such a group we call a cluster. The allowable range of a key set may be thought of as an \( n \)-dimensional hypercube whose \( 2^n \) vertices represent permissible \( n \)-bit keys. A typical key set is some subset of these vertices. As a simplified example, a vertex in a 5-dimensional space may be written as 10111. An immediately adjacent vertex would differ in only one bit position and we say it is one Hamming distance 4 away. For example, 10101 is adjacent to 10111. The vertex 11101 would be two distances away from 10111, and so forth. A cluster in this concept would be a collection of vertices relatively close to each other. Keys belonging to such a cluster are identical in most of their bit positions while the rest are variable. The variable bits are, of course, not necessarily in adjacent positions in the key nor at the low-order end. Two examples of clusters in human readable language are shown in Figure 1.

We propose the criterion that a good address mapping must destroy clusters. This is to say that cluster mates (members of the same cluster) should tend not to be bucket mates (members of the same bucket). In other words, bucket mates should not have their inverse images in the same cluster. We now offer an address mapping using radix transformation to destroy these clusters and to satisfy the solution requirements previously mentioned. The theoretical justification is developed below.

Suppose that a given memory is addressed with a radix \( q \). Usually \( q \) will be either 2 or 10. The number of buckets, \( K \), is equal to \( q^m \) where \( m \) is most naturally an integer. The amorphous binary array is now treated as a number expressed in binary-coded \( p \)-radix digits where \( p \) is prime to \( q \). Radix transformation is then performed on the key to convert it from base \( p \) to base \( q \) using the familiar formula:

\[
A = \sum_{i=0}^{n-1} d_ip^i \mod K
\]

where \( n \) = the number of \( p \)-radix digits in the key array

\( d_i \) = value of \( t^{\text{th}} \) digit

The result is a randomly distributed set of transformations in \( q \)-radix representation, which upon truncation by modulus \( K \), yields the desired addresses \( A \); i.e.:

\[
A = \sum_{i=0}^{n-1} d_ip^i \mod K
\]

For example, let \( p = 11 \), \( q = 10 \), \( m = 4 \), the key be Smith and the machine language be binary-coded decimal (bcd). Suppose 3 bits at a time are grouped to form 11-ary digits. Then the steps \( a, b, c, d \), would ensue.

\[
S M I T H
\]

\[
\begin{array}{ccccccccc}
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\]

\[
\begin{array}{ccccccccc}
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0
\end{array}
\]

\[
\begin{array}{ccccccccc}
2 & 2 & 4 & 4 & 7 & 1 & 2 & 3 & 7 & 0
\end{array}
\]

\[
\begin{array}{ccccccccc}
2,244,712,370_{11} = (5,230,793,172)_{10}
\end{array}
\]

\[
\begin{array}{ccccccccc}
5,230,793,172 \mod 10^4 = 3172
\end{array}
\]

When will any two keys \( X \) and \( Y \) map into the same bucket? Only when they are congruent \( q^m \). We may express this as

\[
X \equiv Y \mod q^m
\]
which implies \( X = Y \pm a q^m \) for some positive integer, \( a \). If this is true when \( X, Y, \) and \( q^m \) are simply magnitudes without radix representation, then the equation holds also for any radix subsequently chosen to express it.

For example, let:

\[
X = |123|, \quad Y = |23|, \quad a q^m = |100|,
\]

where \(| |\) means the magnitude, regardless of what radix is used to express it.

Then:

\[
|123| = |23| + |100|
\]

\[
(123)_{10} = (23)_{10} + (100)_{10}
\]

\[
(146)_9 = (25) + (121)_9
\]

However, consider the difference when \( X \) and \( Y \) are considered initially as amorphous binary arrays which have no attribute of measure until quantified by choosing a radix \( p \) as basis for interpretation. Note now that two arrays quantified in \( p \)-radix, \( (X)_p \) and \( (Y)_p \) (bucketmates), which differ by a magnitude \( a q^m \) are not usually congruent when evaluated in a radix \( p \), where \( p \neq q \). For example, if two arrays are 123 and 023, it can be seen that:

Quantified in radix 10: \( (123)_{10} \neq (23)_{10} \mod |100|
Quantified in radix 9: \( (123)_9 \neq (23)_9 \mod |100|

We choose radix \( p \) relatively prime to \( q \) because \( a q^m \), when expressed in \( p \), has a high percentage of nonzero, non \((p-1)\) digits. (In contrast, when \( a q^m \) is expressed in \( q \)-radix, its representation is \((a)\) followed by all \((m)\) zeros.) This indicates that bucketmates expressed in base \( p \) will differ in many digital positions, which implies that they will differ in many bit positions. In the \( n \)-dimensional space concept, it follows that the inverse images of bucketmates in the key set are relatively distant from each other and hence do not come from the same cluster.

To make this argument more precise, consider the digital configuration of two keys that are made bucketmates by this method. Let \( x_i \) be the \( i \)th digit of \( X \), \( y_i \) the \( i \)th digit of \( Y \), and \( z_i \) the \( i \)th digit of the magnitude \( a q^m \) expressed in \( p \)-radix. Then, from the addition:

\[
y_i y_{i-1} \ldots y_1 + z_i z_{i-1} \ldots z_1 = x_i x_{i-1} \ldots x_1 \ldots z_1 x_1
\]

What are the conditions under which \( x_i \) and \( y_i \) differ? In general, they are different if \( z_i \) is nonzero or non \((p-1)\). In particular, lowest order \( x_i \) and \( y_i \) would be different for all nonzero values of \( z_i \) since no carry is possible from the right. It may be noted further that, if value \((p-1)\) can never occur in \( x_i \) or \( y_i \), an added condition is assured when \( z_i \) and \( z_{i-1} \) are consecutively 0 and \( p-1 \), or \( p-1 \) and 0. This suggests then that in interpreting the binary array of the key, the relationship of \( p \) to \( q \) may advantageously be \( p = q + 1 \). It will be seen later how this relationship is also of benefit in implementation. Examples of \( p-q \) combinations are 3-2, 5-4, 9-8, 11-10, 17-16, and so forth.

To illustrate that \( a q^m \) in \( p \)-radix does indeed have a high percentage of nonzero, non \((p-1)\) digits, an investigation was made of the numbers \( a2^{10} \) expressed in base 3. For each value of \((a)\), a function \( d(a) \) was computed such that \( X = a2^{10} + Y \). Then \( X \) differs from \( Y \) in at least \( d(a) \) digital positions in their radix-3 representation. A frequency tabulation of this function for \( a = 1, 2 \ldots 10,000 \) is shown in Table 1.

<table>
<thead>
<tr>
<th>( d(a) )</th>
<th>Freq.</th>
<th>Cum. Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>100.00%</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>99.93%</td>
</tr>
<tr>
<td>5</td>
<td>197</td>
<td>99.87%</td>
</tr>
<tr>
<td>6</td>
<td>179</td>
<td>97.90%</td>
</tr>
<tr>
<td>7</td>
<td>1046</td>
<td>96.11%</td>
</tr>
<tr>
<td>8</td>
<td>782</td>
<td>85.65%</td>
</tr>
<tr>
<td>9</td>
<td>2573</td>
<td>77.83%</td>
</tr>
<tr>
<td>10</td>
<td>1503</td>
<td>57.10%</td>
</tr>
<tr>
<td>11</td>
<td>2146</td>
<td>37.07%</td>
</tr>
<tr>
<td>12</td>
<td>911</td>
<td>15.61%</td>
</tr>
<tr>
<td>13</td>
<td>510</td>
<td>6.50%</td>
</tr>
<tr>
<td>14</td>
<td>123</td>
<td>1.40%</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>0.17%</td>
</tr>
<tr>
<td></td>
<td>10000</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

This table shows that the radix transformation at least does not map into the same bucket two keys differing only in two bit positions; i.e., having a "distance" of two. In general, however, pair-wise relationships of this sort generally result in stronger affirmation of our thesis for larger key sets, each pair of the set having this relationship. For example, a set of three bucketmates, each pair of which differs
in at least two bit-positions, cannot as a set vary only in two bit-positions.

Table 1 assumes that \( a \) takes on the values 1, 2 \ldots 10,000 with equal likelihood. It is not presumed that every subtle condition which contributes to \( x \) and \( y \) being different has been reflected in the tabulation. The table, therefore, confirms conservatively but strongly the cluster-destroying capabilities of this method of interpreting a key in \( p \)-radix and of transforming it from base \( p \) to base \( q \) to generate an address for a \( q \)-radix memory.

**Hardware Implementation**

To incorporate the proposed algorithm into hardware, one might elect a value for the \( p \)-radix which makes the hardware requirements simple and light. The \( q \)-radix, obviously, will either be 2 or 10, since random access bulk memories are, and most likely will be, addressed in either pure binary or binary-coded-decimal.

It has been noted why \( p \) must be relatively prime to \( q \). Additionally, if \( p \) is chosen equal to \( q + 1 \), then conversion from \( p \) to \( q \) may be performed by the elementary operations of shifting and adding. For example, given a \( q \)-radix address register and a \( p \)-digit \( (d)_p \) to be converted into base \( q \), we would multiply \( d \) by \( q + 1 \). We would implement this in a shift register by adding \( d \) to one register position, then shifting (in effect, multiplying by \( q \)) to the next higher-order position and adding \( d \) again.

We wish to standardize the input so that a high proportion of common hardware may be used whether the device is to be designed for binary address output, for decimal address output, or for both. Partitioning the binary array of the key into 4-bit bytes would be suitable for reasons already explained.

Figure 2 shows a device which would implement the above. It consists of a circulating shift register of \( m \) digital positions \( D_m \ldots D_1 \), a single-digit load register \( D_0 \), a single-digit adder \( H \), and associated logic. The incoming key enters as a serialized train of standardized 4-bit bytes. A byte at a time is loaded into \( D_0 \). Between bytes, the following occurs: The contents of \( D_1 \) and \( D_0 \) are added; the sum is circulated back to \( D_0 \), and if necessary, the carry trigger is set. A right shift follows and another add-shift cycle ensues. This cycle is performed \( m \) times for each byte. The next byte then enters \( D_0 \) and the sequence of operations is repeated. The process ceases when the key array is exhausted.

The finite length, \( m \), of the register automatically truncates the cumulative sum of products at the conclusion of the processing of each byte. The final residue is the desired address. Truncation is valid since the following relation holds:

\[
[(a \mod K + b) \mod K + c] \mod K = (a + b + c) \mod K
\]

where \( a, b, c \) typically are successive bytes and \( K = q^m \). It is apparent that the conversion to any address \( A \) is expressible as:

\[
A = \sum_{i=0}^{n-1} d_i q^i \mod K
\]

where, for a binary array of \( n \) bytes, \( i \) equals 0, 1, 2 \ldots \( n-1 \); and \( d \) is the digital value of the byte.

In the device, the above formula (less mod \( K \)) is advantageously restated in iterative form and implemented as follows:

\[
A = p[p[p[p[\ldots p[p(d_n)\ldots] + d_{n-1}] + d_{n-2}] + \ldots] + d_1] + d_0
\]

This nesting arrangement suggests that the bytes, \( d \), be processed in decreasing order of significance as each byte comes in from the binary array. To generate the address, the formula is evaluated progressively by working
from the innermost nest towards the outermost nest. Within each nest, the following occurs:

- A byte \( d_i \) is brought into \( D_0 \) and added to the product of the radix \( p \) and the current contents of the address register. Since \( p = q + 1 \), this product may be generated digit-by-digit by adding the current contents of \( D_1 \) into both the present and the next higher-order positions.

Thus, the contents of \( D_1 \) participate in two successive add cycles: first when it is in \( D_1 \) and next when it is shifted to \( D_0 \). For example, in an 11-to-10 conversion, if the input byte is 3 and the current contents of the address register are 024, this happens:

\[
\begin{align*}
3 & \rightarrow 44 \\
22 & \rightarrow 267
\end{align*}
\]

For choice of output addresses in either binary or decimal form, logic may be provided as shown in Figure 3. The binary form is, of course, quite straightforward. However, the decimal form requires that the input byte, if it exceeds \( 10 \) in value, be decomposed into a carry and a modulo 11 digit. Furthermore, the influence of the carry must be sustained for the first two add/shift cycles to correct the cumulative results in the address register. For brevity, the detailed logic will not be shown.

An example is presented in Table 2 to illustrate how the binary array of a key having successive byte values of 1, 7, 1, 15 is converted from 11-radix to 10-radix representation; i.e., from \((1724)_{11}\) to \((2204)_{10}\).

It may be noted that if the incoming byte rate of the using system is \( J \), the transformation rate per byte would be \( \frac{J}{(2^m + 1)r} \) where \( r \) is the ratio of the speed of the circuit family employed in the address transformation device and the speed of the circuit family in the using system.

**Empirical Verification**

An extensive empirical testing program was performed on a IBM 704 computer to test the cluster hypothesis and the mapping performance of the radix transformation algorithm. Seven customer key sets were used. In addition, three generated sets of random alphameric patterns were processed. Table 3 shows their characteristics. Finally, a set of values were computed from the theoretical Poisson distribution to serve as criteria of performance. The several test objectives and their results are as follows:

The following notations will be used:

- \( k \) — number of binary positions in memory address register
- \( K \) — number of buckets in the memory = \( 2^k \)
- \( C \) — bucket capacity in records
- \( N \) — number of keys in key set
- \( b \) — bucket occupancy level (stochastic variable in frequency distribution)

\[ \bar{b} = \text{average number of records per bucket} = \frac{N}{K} \]

\[ M = \text{number of record spaces in memory} \]

\[ L = \text{load factor} = \frac{N}{M} = \frac{N}{KC} = \frac{b}{C} \]

\[ R = \text{percent of } N \text{ lodged in home buckets} \]

\[ e = \text{underflow or overflow in } i^{th} \text{ bucket} \]

\[ g = \text{net undisposed overflow at } i^{th} \text{ bucket} \]

\[ f = \text{number of buckets having occupancy } b \]

**Figure 3. Block Diagram for Binary/Decimal Address-Generating Device.**

From the collection of the Computer History Museum (www.computerhistory.org)
A basic test is that the transformation maps a typical key set into an address set essentially as evenly as a randomly distributed key set. This would affirm the cluster-destroying effect of the algorithm. The basic test is extended by determining whether the technique results in an efficient seek factor. This factor is the number of separate access-mechanism movements required to locate a record, averaged over the total number of records in an application. It measures the combined performance of the mapping algorithm and of the overflow technique, and should be closely comparable to that of a randomly distributed set using the same overflow procedure. The overflow technique is of secondary interest here but, since one has to be assumed to develop the values for the seek factor, the “consecutive spill” routine was used. If a record is not found in its home bucket, consecutively higher-numbered locations are searched until it is found.

To establish universality, the algorithm should map key sets of widely divergent structural characteristics into a given memory configuration with essentially equal effectiveness. A given memory configuration is one in which
the relevant parameters are pegged at certain values. These parameters are average bucket occupancy, \( b \) (i.e., \( N/K \)), and load factor, \( L \). "Equal effectiveness" means that essentially the same percentage, \( R \), of the entire key set gets housed in home buckets. For example, for \( b = 64 \), and \( L = 95\% \), a large data file using this algorithm can expect to have, say, 97\% of its members housed in home buckets. This yardstick is meaningful and directly useful to the application planner.

To affirm code independence, the addressing algorithm must map essentially the same percentage of a key set into home buckets when the key set is expressed in different binary-base codes. On Table 3, three of the sets, \( D, F, \) and \( H \), are expressed in both binary-coded-decimal and 2-out-of-5 notation to test this independence.

A 17-16 radix transformation was performed on all twelve sets. For each key set, the value of \( K \) was assigned seven different values: \( K = 2^k \), where \( k = 1, 2 \ldots , 7 \); i.e., \( K \)-values of 128, 256, 512, 1024, 2048, 4096, and 8192.

A frequency distribution was tabulated relating \( f \) to \( b \) for each \( K \). From this distribution, \( R \) was determined for a load factor of 100\%, by choosing \( C = b \). Other values of \( R \) were computed for lower load factors. The latter were obtained by successively increasing the value of \( C \), since

\[
L = \frac{b}{C}.
\]
The results from the IBM 704 computer simulation were plotted in Figure 4 as a curve with $R$ against $L$. A family of curves was thus generated with the parameter $b$ taking on different values. The results for $R$ for a given $b$ and $L$ were so nearly identical (root mean square deviation less than 1%) for all nine data key sets that only one curve representing their average was plotted for each value of $b$. The $R$ values for $b$ of 4, 2, and 1 were similarly uniform for all nine sets but are not shown in the graph because they offer only two or fewer points in our range of interest of the load factor; i.e., 80% to 100%.

The results for the three originally random key sets were also averaged and plotted as a separate group in Figure 4. These $R$ values also show an rms deviation of less than 1% related to its own group and to the entire twelve sets.

The results for three random key sets whose address mapping was performed by $k$-bit truncation rather than by our algorithm are also plotted in Figure 4. Again the rms deviation, among themselves and related to the twelve-key-set average, is less than 1%.

A set of theoretical values for $R$ were computed from Poisson Distribution Tables, using the same values for $b$ and $L$ as in the empirical runs. The relationship which yields $R$ is simply:

$$R = \frac{\sum_{b=0}^{\infty} p_b \cdot b + C \sum_{b=C+1}^{\infty} p_b}{b}$$

where $p_b$ is the Poisson probability for an occupancy level of $b$ records per bucket, and $C$ is the capacity of a bucket adjusted to get a specific load factor $L = \bar{b}/C$.

Since all curves were quite close together, it was decided for clarity to graph only the curves for the empirical results without depicting their points. For comparison purposes, the theoretical Poisson points were shown but without their curves.

The average seek factors, $S$, were calculated for the nine data key sets; their arithmetic means for a given pair of parameters (bucket capacity and load factor) are plotted in Figure 5. The same operations were performed on the three random key sets.

The seek factor was arrived at thus:

$$s = \frac{K + \sum g_i}{K}$$

CONCLUSIONS

The empirical results and the hardware implementation indicate that the functional objectives sought have indeed been met.

The empirical results show that for a given set of parameters $b$ and $L$, the percent $R$ of the data set lodged into home buckets tends towards a central value with narrow deviation. These
central values were shown to be quite close to the theoretical Poisson values. This suggests then that a comprehensive table of theoretical values of $R$ as related to $b$ and $L$ may be set up for the use of the application planner. Using this table to make reasonable predictions for the mapping performance of the radix transformation algorithm, he could attempt systematic trade-offs between the parameters and the performance until he is satisfied with the combination.

The simulation effort with widely differing key sets shows this algorithm to be a directly usable method that is independent of key length, source language and machine code.

Key addressing by radix transformation meets its optimal fulfillment when implemented as independent hardware with a binary address output. Augmented to provide the added option of decimal address output, it is slightly less simple but offers wider flexibility in system applications.

ACKNOWLEDGEMENTS

The author is indebted to the following individuals for auxiliary participation in the work covered by this paper: Mr. R. F. Arnold for major contributions in the early evolution of the transformation algorithm; Mr. F. Magness for assistance in the detailed logical design; and Mr. R. Togosaki and Mr. J. Barnes for preparing the several 704 programs for computer simulation and verification with customer key sets. The latter were made available through the courtesy of several IBM Sales and Service Bureau offices.

REFERENCES