G. A. Blaauw

Indexing and Control-Word Techniques*

Abstract: In large-scale computers the details of data handling, such as indexing, transmission and ordering, may be performed either by programming or by built-in machine operations. An analysis of the most frequently performed functions justifies the expansion of single-valued index quantities to three-valued control words and the specification of built-in increment, count and refill operations to be used with these control words. STRETCH, the large-scale computer which is being developed by IBM for the Los Alamos Scientific Laboratory, provides these control-word functions for data-handling operations.

Introduction

One of the basic requirements for a computer is that writing the program for a calculation take less effort than performing the calculation without the computer. This requirement can be satisfied when the calculation permits a program to be repeated with different sets of data. In the earliest machines the technique employed was to change the contents of storage locations between successive executions of the program. A later method of achieving the same result was to change the addresses used by the program in referring to data, rather than changing the data at a given address. This procedure widened the scope of computer applications considerably. Early computers, whose programs were specified by pluggable wiring, paper tape or cards, permitted little or no address alteration. The invention of stored-program computers provided a major advance because it allowed a program to be treated as data, so that any instruction of a program could be modified by the program itself. The main application of this general facility was for the modification of addresses. Subsequently, it became apparent that programmed address computation, though sufficient in theory, was cumbersome in practice. Too much computing time and program space were required to perform these auxiliary operations. A remedy was provided by an address register, also called *index register* or *B-line*,[1] whose contents could be added to the operand address. In recent machines, several index registers—up to one hundred—have been made available.

The historic development outlined above shows that address computation has partly taken the place of data transmission and has subsequently been simplified by the introduction of index registers. Providing machine functions, such as indexing, for operations which could be programmed was not new since in theory all machine instructions but one are redundant. That is, an instruction repertoire can be replaced by one single, well-chosen instruction.[2] In practice, a repertoire of more than one instruction is justified by the operating time and program space which is saved. Similarly, special-purpose registers like index registers may be justified when they increase the effective speed and capacity of the computer. The gain in performance then offsets the expense of the added equipment, improving the performance/cost ratio. This type of performance gain normally is accompanied by greater programming ease. Programming ease greatly affects the form which an added function should take, but, because it is hard to express in a cost figure, it is rarely used as the sole justification for added equipment.

In the design of the STRETCH Computer,[3] an attempt has been made to achieve great flexibility and generality in machine functions. The indexing functions and the associated instruction set, consequently, were examined carefully. The general principles which were considered in this examination will be discussed first. The built-in functions which were developed for the STRETCH Computer as a result of the examination will be described subsequently and illustrated by examples.

Index function

Index functions may be divided into four groups: address modification, index arithmetic, termination, and initialization. The first group is used in addressing operands and provides the justification for the existence of index quantities. The other groups concern the task of changing the index quantities, the tests for end conditions and the set-up procedures. These operations are often termed *housekeeping*.

* A condensed version of this paper was presented at the Eastern Joint Computer Conference, December 1958, under the title, "Data Handling by Control-Word Techniques."

The common use of an index register is the addition of its contents, the *index value,* to the address part of an instruction, which will be called the *operand address,* in order to address memory with the sum, the *effective address.* This operation is called *address modification.* The operand address and the index value remain unchanged in storage in this operation.

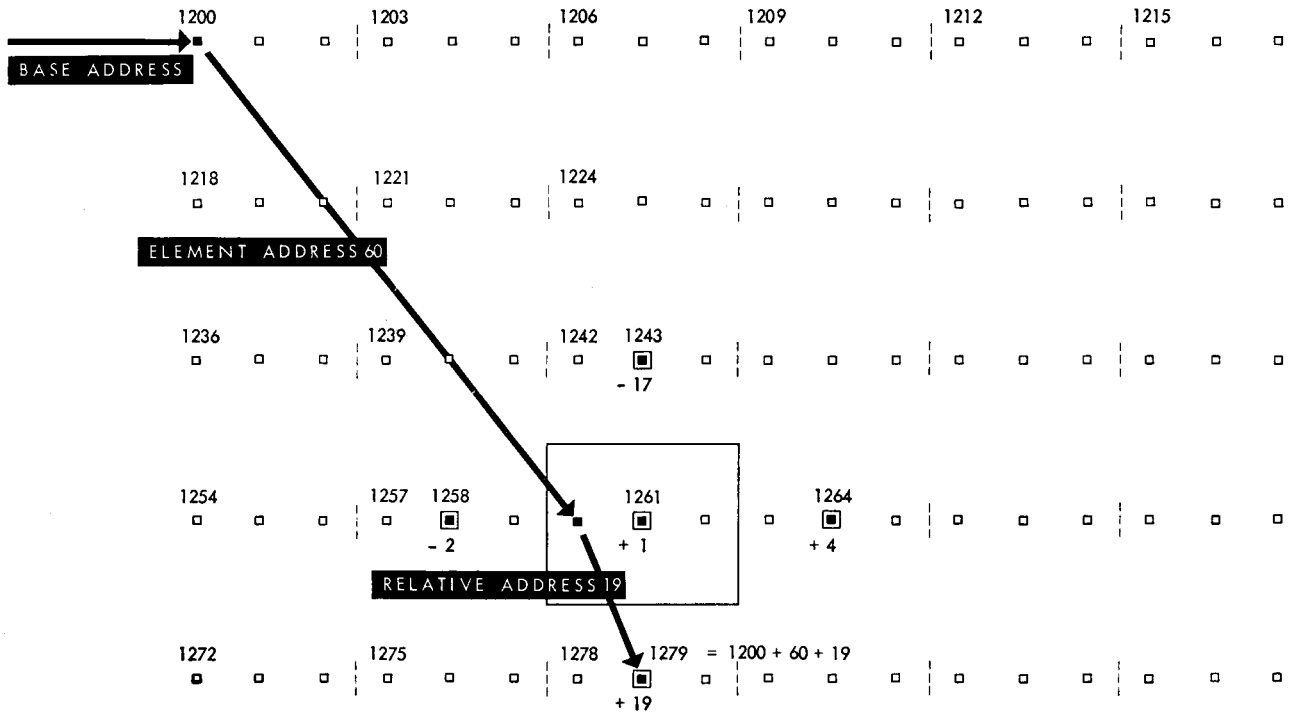
Address modification is used in general to address successively the elements of an array. An array may be one dimensional, or multi-dimensional, and its elements may be single-valued or multiple-valued. The address of a value which is part of an array can be subdivided into three distinct parts. The first part, the *base address,* identifies the location of the array within memory. The second part will be called the *element address.* This address concerns the location within the array of the element which is currently used in the computation. The element address is specified relative to the base address and is independent of the location of the array in memory. The third part of the address of an array value is the *relative address* which specifies the location of the array value relative to the current element. The relative address is independent of the location of the array or the selection of the current element. The array value may be part of the current element or it may be part of another element. A well-known case in technical computation is the addressing of right, left, upper and lower neighbors of an element in a two-dimensional array. Figure 1 illustrates this case and shows how the address of a particular array value is formed as the sum of base address, element address, and relative address.

The base address and relative address are constant throughout the execution of the program. The base address is determined as part of the task of memory allocation. The relative address is determined as part of the programming task by the characteristics of the computation to be performed. The element address, on the other hand, is not constant. It changes as the computation proceeds from one element to the next.

The three components, base, element and relative address, must be available during address modification. Therefore each of these addresses must be found either in the operand address part of the instruction or in the index values of index registers. In order to allow effective address modification, the variable part of the array address, the element address, should be part of an index value. The relative address is used to address different values for a given element address. In order to preserve the identity of the selected element, the index value, which contains the element address, must remain unchanged. Therefore, the relative address should be part of the operand address. The base address may be part either of the operand address or of an index value. In the first case, it is added to the relative address; in the second case, it may be added to the element address.

As a computation proceeds, successive elements of an array are addressed. The element addresses are generated by the algorithm which is appropriate for the use of the array in the computation. Since the element address is part of the index value, the address computation may be accomplished by *index arithmetic.* In a large number of cases, the algorithm used is a simple recurrent process

Figure 1 **90 storage locations used for three-valued, two-dimensional array of 6×5 elements. (Relative addressing shown for second value of an element and its four neighbors.)**



in which a new index value is obtained by the addition of an *increment* to the old index value.

There are several algorithms which cannot be described by a simple incrementing process. In particular, some algorithms make use of variables which are data or instructions rather than known parameters of an array. The use of data in index arithmetic occurs in *table reference* techniques. The use of instructions in index arithmetic occurs in *indirect addressing*. In this mode, the effective address is used, not as the address of an operand, but as the address of an instruction whose effective address is the address of the operand.

The conventional use of the effective address as the operand address is called *direct addressing*, in contrast to the indirect addressing mode. In a simple incrementing process, another addressing mode, *immediate addressing*, is often used. In this case, the effective address is used as an operand, rather than as the address of an operand.

Each time an index is altered by index arithmetic, a test may be performed to determine when the last element of the array is addressed. This process is called *termination*. Some of the forms of the test are: limit comparison, length subtraction, and counting. In *limit comparison*, the current index value is compared with a given constant, the *limit*. In *length subtraction*, a given variable, the length, is reduced by the value of the increment and tested for zero. In *counting* a given variable, the *count* is reduced by one and tested for zero. The three methods of test are closely interrelated. When the base address is part of the index value, the limit is the sum of base address and length. The length, in turn, is the product of increment and count. Counting permits the test for completion to be independent of base address and increment, such that even an "increment" of zero is possible.

Instead of using a separate value such as limit, length, or count, the index value itself can be used to determine the end of the process. In that case, the index value serves as a length, and a limit of zero is implied. This approach requires a minimum of information and is followed in the IBM 704, 709, and 7090. A greater degree of freedom in specifying index values and tests is, however, very desirable. Therefore, independence of index value and test for termination is preferred. In the STRETCH Computer, counting has been chosen as the primary means for determining the end of an index modification sequence. However, the conclusions reached in the course of the discussion are equally valid when a limit or length is used.

After the last element of the array is addressed, the index value and count must each be changed to the initial setting for the array to be addressed next, which may be the same array or another one. This housekeeping operation is called *initialization*. Of course, initialization also occurs prior to the entire array-scanning operation. This case is the least frequent and is usually part of more general loading and resetting procedure. For these reasons its characteristics influence the indexing procedures to a lesser degree.

A summary of the index functions which have been described is shown in Table 1. The quantities which occur in the indexing procedure for a simple array are listed in the second column. The operations which make use of these quantities are listed in the third column.

Table 1 **Summary of index functions.**

Function	*Quantity*	*Operation*
Index Use	Index value	Address modification
Index Change	Increment	Incrementing
Index Test	Count	Counting and zero testing
Index Reset	Next initial:	Replacement of:
	Index value	Index value
	Count	Count

Of the quantities listed, the index value is in the index register. This leaves four quantities which must reside somewhere. Earlier approaches have relied on storing these quantities in general memory locations. Of the four operations listed, only address modification is usually performed as a built-in machine operation. In most earlier machines the other three operations are performed by standard arithmetic instructions. In the following sections, the possibility of storing more quantities in the index register and providing more built-in operations will be considered.

Instruction format

A systematic method of operand addressing requires a uniform means of address modifications for all operands. Relative addressing requires at least one field for direct operand designation, called the *operand address* field, and one field for indirect operand designation, called the *index address* field. The latter field specifies the address of the index used in address modification. Providing more direct address fields for each operand serves no purpose. More indirect address fields would be infrequently used. They would find application when an index arithmetic algorithm is used which forms the sum of two or more independently computed index values, called *multiple indexing*. In order to provide for this case, it was chosen not to burden the operand designation with added index address fields, but to provide a separate instruction, LOAD VALUE WITH SUM. This instruction adds any selected number of index values, and places the sum in another selected index value. This procedure of providing for operations whose application is important but not frequent, by means of specific instructions rather than by fields which appear in all instructions, has been followed in all applicable cases. As a result, the instruction information content is improved, since one out of many codes is used, rather than an extra bit for each code. Also, the efficiency with which a program can be stated is improved, since the infrequent use of an extra instruction is easily offset by the greater information content of each frequent instruction, while on the other hand, omitting the instruction entirely would require a subroutine each time the need arises. Another example of this procedure is the instruction LOAD VALUE EFFECTIVE,

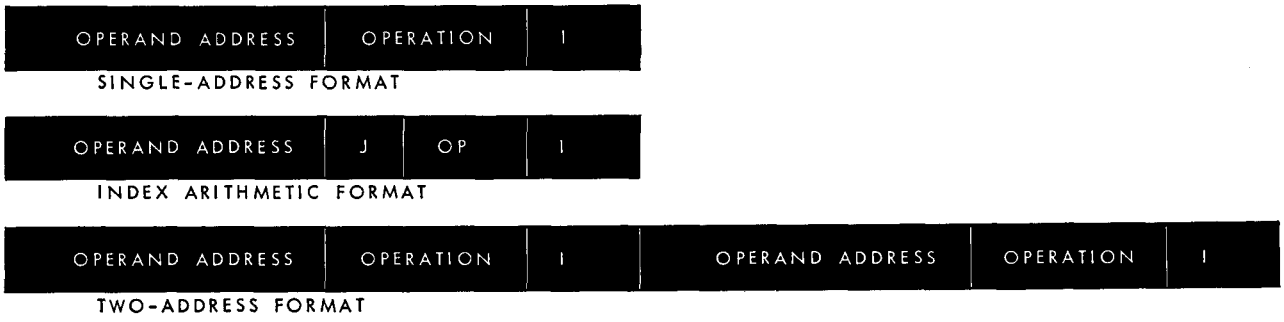


Figure 2 **Instruction formats.**

which gives the equivalent of indirect addressing by loading an index value with the effective address of the instruction at the addressed memory location.

With an operand address field and an index address field required to specify each operand and with several operands necessary for most operations, the instruction format would become inefficient unless implied addresses or truncated addresses are used.

In arithmetic operations, the accumulator usually is used as the *implied address.* An add-type operation, for instance, may have one implied operand in the accumulator to which an explicitly specified operand is added. The sum replaces either the implied operand or the specified operand. Of the three addresses required by the operation, only one is stated explicitly. This gain in efficiency is nullified when the ADD is preceded by a LOAD and followed by a STORE. Therefore, implied addresses provide a gain in instruction-bit efficiency only when repeated reference is made to the implied address without intermediate change of the implied operand. In arithmetic operations, repeated reference to implied addresses occurs with sufficient frequency to justify the single address instruction format outlined below. In index arithmetic operations, the use of implied addresses has been extended by specifying more than one operation in one instruction, as will be described in the following sections.

A second means of reducing the number of bits required for operand specification is the use of a *truncated address.* The truncation of the address reduces the number of available address locations, and consequently makes the instruction set less general. A truncated address for index registers may be justified, however, because a limited number of index registers is usually used in a program and a complete address would therefore be inefficient. A second justification is that limiting the number of index registers permits preferred treatment for these registers to speed up index arithmetic operations and address modification. A third justification is that a truncation of the index address makes it possible to include a second index address in index arithmetic instructions, which greatly improves the efficiency of these instructions. Nevertheless, some applications require complete generality for index addresses. For these cases, an instruction RENAME effectively expands an index address to the full capacity. The instruction loads an index register from any desired memory location, retaining the address of the memory location; the contents are automatically stored back at the original location before the index register is loaded by a subsequent RENAME instruction.

Another possibility for improving the efficiency of operand specifications is the use of a truncated operand address. This method was not used, however, since the size of relative addresses would be restricted and the base address could not be part of the operand address.

As a general pattern, a *single-address* instruction format is used in the STRETCH Computer. The name "single address" refers to the operand address and ignores auxiliary addresses such as the index address. The format permits specification of operations which require only one explicit address. The index address field, *I,* is used in address modification and is part of the operand specification. Index arithmetic instructions use the *index arithmetic* format which is the single-address format to which a second index address field, *J,* has been added so that the second operand can be addressed explicitly. Some operations, for which two complete explicit operand addresses are desired, use a *two-address* format. This format consists of two single-address formats and has double the length of the single-address format. Figure 2 shows three basic formats which are used.

Increment instruction

Index incrementing could be performed in the accumulator by a series of three single-address instructions which add the increment to the index value and return the result to the index register. Actually, only the increment and the subject index need specification, and since the index address is truncated, the index arithmetic format can be used to specify the entire operation. Such an ADD TO VALUE operation can make use of the index adder which is provided for address modification. The main arithmetic process for data is then separated from the housekeeping process. Data registers need not be altered. Because of these advantages, an ADD TO VALUE operation is normally provided when index registers are available. In the index arithmetic format which is used in the STRETCH Computer, the operand address specifies the address of the increment. The operand address can itself be indexed just as any other operand address. This gives indexable index arithmetic.

The quantity used in incrementing, the *increment,* is specified explicitly in the increment instruction. A different approach is possible. The increment could be associated with the index, such that the address of the increment is known whenever the index is addressed. The increment is then specified by an implied address. As was pointed out before, an advantage is obtained by implied addressing when the increment remains unchanged. Furthermore, such an ADD TO VALUE operation should be combined with another operation which uses the same index address. For instance, it would be possible to specify in one single-address instruction the use and subsequent incrementing of an index. This method, however, loses its value when several increments must be used to change an index value, or when the incrementing and index use must occur in different parts of the program. In order to achieve greater generality, a separate ADD TO VALUE instruction has been chosen in preference to a combined instruction. Several variations of the basic ADD TO VALUE instruction, permitting sign inversion and immediately addressing, are available.

Count

In the termination of array scanning, more than one count may be used, just as several increments may be used in index arithmetic. Most frequently, however, a single count is used. It therefore is profitable to associate the count used in the termination with the index value to which the process applies, and use implied addressing. Since counting normally occurs when the index value is changed, it is logically consistent to specify incrementing and counting in one index arithmetic instruction, ADD TO VALUE AND COUNT. This instruction is available in addition to ADD TO VALUE. It becomes equivalent to "count" when the increment is zero.

An implied address for the count can be obtained in various ways. A solution, economical in time and space, is to place index value and count as separate fields in one word. Such a word will be referred to as a *control word.* The instruction ADD TO VALUE AND COUNT adds the addressed increment to the index value, reduces the count by one and provides a signal when the count becomes zero.

The choice of counting as a test for termination and the use of an implied address for the count does not preclude other termination tests. In particular, a COMPARE VALUE instruction is made available to allow limit tests and an ADD TO COUNT instruction can be used for the equivalent of length subtraction. These instructions add flexibility to the instruction set but they are less efficient than ADD TO VALUE AND COUNT.

The following example, to be expanded later, illustrates the use of counting in a simple technical computation. It is required to multiply Vectors A and B. Each vector has n elements. Vector A has its first element at a_0, Vector B has its first element at b_0. The product is to be stored at c_0. A is stored in successive memory locations. B is a column vector of a matrix whose rows have p elements and are stored in successive memory locations. Therefore, the elements of B have locations which are p apart. The program is shown in Table 2. Multiplicand and multiplier are specified in Instructions $f+3$ and $f+4$. Their product is added to the accumulator content, which contains the sum of the previous products. This operation is called *cumulative multiplication.* The count in control word i terminates the cumulative multiplication. The count in control word j is not used. The example shows that the use of the control words i and j in two instructions requires five added instructions in order to change, test, and initialize these control words. Three of the latter instructions are in the "inner loop." Even though the simplicity of the arithmetic process tends to over-emphasize the housekeeping burden, further simplification of the indexing procedure would be desirable.

Advance

An array in which elements have adjacent addresses, such as Vector A in Table 2, requires an increment of one. The frequency of occurrence of an increment of one suggests the definition of an "advance and count" operation which is an ADD TO VALUE AND COUNT operation with an implied immediate increment of one. The advance operation then can be combined with another single-address operation. A suitable candidate is the conditional branch operation which refers to the zero-count test. The new instruction then becomes ADVANCE, COUNT, AND BRANCH. Several variations on ADVANCE, COUNT, AND BRANCH can be and have been provided, but they add no new indexing concepts and consequently will not be discussed in detail.

In the example of Table 2, Instructions $f+6$ and $f+7$ can be replaced by one ADVANCE, COUNT, AND BRANCH operation.

Progressive indexing

In discussing index use, it was pointed out that a base address can be part of the operand address or of the index value. When the base address is part of the index value and the relative address is zero, the operand address is not used at all. The operation therefore can be combined with an ADD TO VALUE AND COUNT operation. The index value is first used as an effective address to address memory and subsequently incremented by the operand address, which acts as an immediate increment. This order of events occurs also when two separate instructions are used. The operation part of the instruction, besides specifying the arithmetic operation, also specifies: *Use index value as the effective address and subsequently increment and count.* This type of indexing will be called *progressive indexing.* Simple arrays which permit progressive indexing occur both in data processing and in technical computations.

In the vector-multiplication problem of Table 2, the base addresses a_0 and b_0 could be placed in the value field of i_0 and j_0, respectively. If progressive indexing were used, Instruction $f+5$ could be combined with $f+4$ and, instead of using the ADVANCE operation, Instruction

Table 2 **Vector multiplication, using COUNT.**

• Instructions

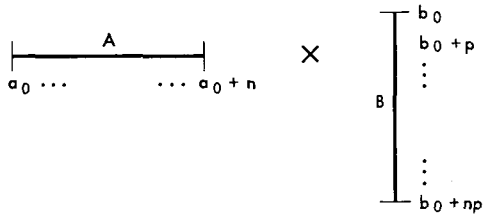
Initial setup	f	Load i from i_0
	$f+1$	Load j from j_0
	$f+2$	Set accumulator to zero
Vector multiply, inner loop	$f+3$	Load cumulative multiplicand from a_0, indexed by i
	$f+4$	Multiply cumulatively by b_0, indexed by j
Housekeeping, inner loop	$f+5$	Increment j by p
	$f+6$	Increment i by 1, count
	$f+7$	Branch to $f+3$ if count did not reach zero
Vector multiply, outer loop	$f+8$	Store cumulative product c_0

• Control Words

Contents after executing the inner loop x times

Address	*Index Value*	*Count*
i	x	$n-x$
i_0	0	n
j	xp	.....
j_0	0	....

• Diagram of vector dimensions.



$f+6$ could be combined with $f+3$. As a result, the program is shortened both in instructions and in execution.

The use of progressive indexing in a data processing operation is illustrated in Fig. 3. A series of elements of different length is processed. As part of the computation which is appropriate for an element, the element is addressed, using progressive indexing. As a result, processing can proceed from one element to the next without added index arithmetic. The example shows the use of indexing words and bits within a word, as provided in the STRETCH Computer.

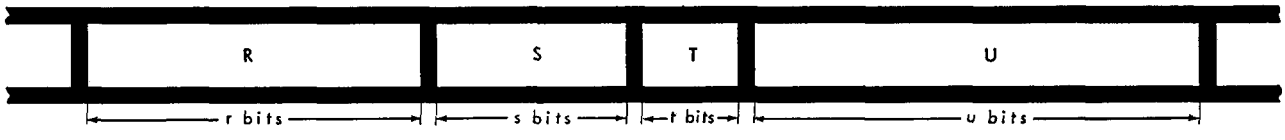
Data transmission

When an increment of one is implied, as discussed in the case of the ADVANCE operation, the count becomes the equivalent of a length and represents the number of adjacent words in the addressed memory area. When, furthermore, the index value is used as an effective address, as in the case of progressive indexing, the initial index value is the base address, and addresses the first word of the memory area. A memory area can, therefore, be specified in position and length by the value field and count field of a control word. This makes it

Figure 3 **Progressive indexing of elements with varying length.**

• Instructions

e	Load element R, length r bits, from location specified by i and increment i by r.
$e+1$	Compute with element R.
$e+2$	Load element S, length s bits, from location specified by i.
$e+3$	Compute with element S.
$e+4$	Store new element S, length s bits, at location specified by i and increment i by s.
$e+5$	Add one to element T, length t bits, in location specified by i and increment i by t.
$e+6$	Load accumulator with a constant.
$e+7$	Compare accumulator to element U, length u bits, in location specified by i and increment i by u.



convenient to specify the memory areas involved in data transmission by means of control words and gives the control word the characteristic of a shorthand notation for a memory area.

Data may be transmitted between two memory areas or between input or output units and memory. The data which are transmitted in one operation will be called a *record*. A control word may be used both for indexing and data transmission. This generality makes it possible to associate a control word with a record and use it to identify the record throughout an entire program, including reading, processing, and writing.

The use of control words in transmission instructions is particularly convenient when data can be moved directly between input-output units and general-purpose memory. This ability is incorporated in the STRETCH Computer as well as in other recent computers, and avoids special-purpose areas to buffer records. An input-output instruction specifies the input-output device used, the operations to be performed and the address of a control word. The two-address instruction format is used. Data can move directly between the device and the memory area specified by the control word.

Data ordering

A common procedure in data-ordering operations, such as sorting, merging, queuing, inserting, and deleting, is to move records from one memory area to another. With control words it is possible to replace the transmission of a record containing many data words by the transmission of a single control word which specifies that record.

As an example, consider n records stored in random order. It is desired to write the records on tape in proper sequence. The sequencing is accomplished by ordering the control words which are associated with the records. The "key" value of each record is addressed relative to the base address in the control word for that record. By comparing the key values of the records, their proper sequence is determined. In the course of this procedure the control words may be placed in the correct order in successive memory locations. The sequence of the control words then specifies indirectly the sequence of the associated records. When the records are written on tape, the control words are used in the order of their addresses. Consequently, the records appear on tape in the desired sequence. No record transmission is required other than from memory to tape.

The preceding example illustrates the case of a series of records which are to be processed as a *group*. The records cannot be described by a single control word since they are not in successive memory locations. The group is described by a series of control words. The transmission to or from input-output devices can, however, be mechanized by defining a *chain* of control words. The chain is started by the control word specified in the instruction. The chain is continued by taking control words from successive memory locations. The chain is ended when some kind of end condition is sensed. A convenient end condition is the presence or absence of a bit in the control words. This bit will be called the *chain bit*. Thus, a single input or output instruction can, by means of a chain of control words, initiate the transmission of a group of records. Records which appear in memory in random order are said to be *scattered*.

Control words were introduced in the IBM 709 in order to permit grouped-record transmission to or from external devices. In the IBM 7070, control words can be used both for grouped-record transmission and for indexing. Both machines establish a chain of control words by placing the words in consecutive memory locations.

An example of data ordering is the case of the deletion of one record from a group of records. Assume the records $A \ldots Z$ are in consecutive memory locations. To delete Record D from this series, the records $E \ldots Z$ would have to be moved to the locations previously occupied by $D \ldots Y$. The use of control words greatly simplifies this procedure. The grouped records can be in random order in memory with their order established by control words which are in consecutive memory locations. The deletion of Record D is accomplished by removing its control word from the table of control words and moving all subsequent control words one space such that they again form a continuous table. Table 3 illustrates this procedure. The insertion of a record in a group of records may be handled by reversing the process.

Table 3 **Sequence of control words.**

Old	*New*
A	*A*
B	*B*
C	*C*
← *D*	*E*
E	*F*
F	*G*
•	•
• ↑	•
•	•
•	•
X	*Y*
Y	*Z*
Z	

Some conclusions may be drawn concerning the use of control words in data transmission and data ordering.

First, since record transmission is replaced by control-word transmission, an advantage in storage space and transmission time is achieved. The advantage of the procedure is dependent upon the size of the record. When the record is one word long, it is more advantageous to transmit the records.

Second, the location of a record and its control word are independent, which facilitates data ordering by control-word manipulation.

Third, the use of identical control words for both in-

dexing and data transmission simplifies data ordering operations.

Fourth, the records can be scattered in memory. However, the control words have their sequence indicated by the sequence of their memory addresses. As a result of this restriction, activity on one record may require relocation of several control words.

Refill

The advantage of using control words in data handling is increased when control words as well as records can be scattered. Random addresses for control words imply that a means for specifying their sequence must be provided. A straightforward solution has been found by introducing a *refill* field in the control word which specifies the memory address of its successor. The control word then contains three fields: the value field, the count field, and the refill field, as shown in Fig. 4. This solution is particularly attractive since it also completes the indexing requirements stated in Table 1. It was shown at that point that an indexing operation required specification of: index value, increment, count, and next initial index value and count. All these quantities except the last two have been specified so far, either in instructions or in the control word. The last two quantities can now be specified by the refill address. This address can refer to a second control word, whose value and count field specify the next initial setting. In fact, the second control word is the next initial control word. The refill field then serves the general purpose of linking a control word with the next control word to be used.

The operations which use the quantities mentioned above were listed in Table 1 as: address modification, incrementing, counting and zero testing, replacement of index value and count. All these operations, except for the last, have been specified as machine functions. The last operation can be restated as: *Replace the index word by the word at its refill address location.* The operation as stated makes use of an implied address. Therefore, the operation can be part of an INCREMENT, COUNT, AND REFILL instruction. This combination of operations is only meaningful when the refill operation is conditional. An obvious condition is that the count should reach zero. In addition, the instruction repertoire can include other instructions, such as an unconditional operation REFILL.

The refill operation can also be incorporated in input-output data transmission control. The control words comprising a data transmission chain need no longer be located in successive memory locations. One control word refers to the next through its refill address. The chain bit indicates the termination of the chain and hence stops transmission.

The refill function requires that the refill address be part of the index word. When a computer word is not large enough to contain all three fields, a partial solution can be found by using two adjacent words in memory. This procedure has been used in the input-output control of the IBM 709. In a series of control words in that machine a word may be placed which has the character of the instruction: *Continue with the word at the specified location.*

An alternate use of the refill address has been considered. The refill address could be used as a branch address rather than a control-word address. Whenever the test condition is satisfied, a branch is made to a subroutine which takes care of all termination and initialization procedures. As a minimum, the control word could be reloaded, but more elaborate programs could be performed. The procedure is more general than the refill operation as defined above. The cost of this generality, however, is loss in efficiency in the case of the minimum reload procedure; a branch as well as a load operation is performed; each control word requires an associated load instruction. In other words, the use of an implied address in the main program is obtained at the expense of explicit addresses in a subroutine. The ability to permit more elaborate initialization procedures is often incompatible with the use of the control word in different parts of a program. For these and other reasons, the refill operation has been preferred over the branch procedure or one of the many variations thereof.

Index applications

The basic indexing formats and functions have been defined in the preceding sections. The use of this mechanism will be demonstrated in the remainder of this paper by re-examining the examples which were used above in illustrating the evolution of the mechanism, as well as by considering some more elaborate applications. Of the indexing applications, the simple example of vector multiplication described earlier and its expansion to matrix multiplication will be discussed.

The vector multiplication program was listed in Table 2. The same program using the refill operation is shown in Table 4. The control words are automatically reset. When the program is executed repeatedly, it is sufficient to start at the initial setup instruction *g*. When however, the execution of the program is stopped prematurely and a restart is required, the preparatory steps *g*-2 and *g*-1 are required, which load *i* and *j*. Thus, loading of *i* and *j* should always be part of the program load procedure. The control words *i* and *j* are specified by

Figure 4 **Control-word format.**

truncated addresses and are located in the index registers. The control word i_0, however, has a complete address and can be located anywhere in memory. The program illustrates the use of an ADVANCE, COUNT, REFILL AND BRANCH instruction. Because the base addresses a_0 and b_0 are part of the operand address, the control word i_0 can serve as a refill word for both i and j.

The program for matrix multiplication is shown in Table 5. Index i describes the row element of Matrix A. The index repeats the same row p times. Index j is used to address the elements of a column of Matrix B. It is incremented n times by p, then reloaded from j_0. The count of j is not used. At the end of every vector multiplication, j_0 is incremented by 1, thus selecting the next column vector of the multiplicand.

The incrementation of j_0 is counted p times and used to determine the end of the product row. Index k is used to determine the end of the entire matrix multiplication.

The program shows that a reasonably complex indexing procedure can be described satisfactorily and compactly. The following observations can be made:

(1) Only Instructions $h+6$ and $h+11$ contain constants which describe the locations and dimensions of the matrices. Both instructions could use a direct address instead of an immediate address, however. In that case, the program is independent of the data. The use of a direct address slightly increases the execution time.

(2) The constants describing matrix locations and dimensions appear as single quantities in instruction and control-word fields. The only exception is the constant m which appears as part of the product mp. Note that only control words i_{00}, j_{00}, and k_0 should be supplied by the programmer. All other control words are developed during program execution or preliminary setup.

(3) The automatic refill is used in the inner loops. The refill operation is supplemented by load operations in the outer loops. The refill operation is no substitute for preparatory operations required for restart procedures.

Record-handling applications

Record-handling techniques have application both in technical computation and data processing. The examples to be discussed are a read-process-write cycle, ordering, and a file-maintenance procedure.

The use of control words for a simultaneous read-process-write cycle is illustrated in Fig. 5. Here $X-x$ describes a control word, which, by its value and count fields, defines memory area X and which has the address x in its refill field. Location x contains the next control word in the chain, $Y-y$, defining Record Y. Control word $Z-z$ is placed at Location y. Because control word $X-x$ is stored at Location z, a "ring" of three memory areas, X, Y, and Z is set up in which X is followed by Y,

Table 4 **Vector multiplication, using COUNT and REFILL.**

• Instructions

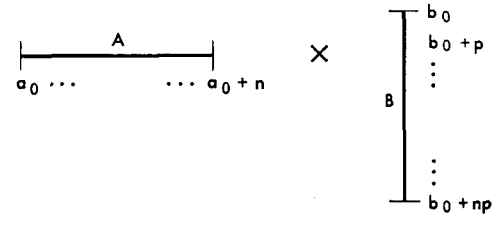
Preparation	$g-2$	Load i from i_0
	$g-1$	Load j from i_0
Initial setup	g	Set accumulator to zero
Vector multiply, inner loop	$g+1$	Load cumulative multiplicand from a_0 indexed by i
	$g+2$	Multiply cumulatively by b_0 indexed by j
Housekeeping, inner loop	$g+3$	Increment j by p, count, refill when count reaches zero
	$g+4$	Advance i, count, refill when count reaches zero, branch to $g+1$ when count does not reach zero
Vector multiply, outer loop	$g+5$	Store cumulative product at c_0

• Control Words

Contents after executing the inner loop x times

Address	*Index Value*	*Count*	*Refill*
i	x	$n-x$	i_0
i_0	0	n	i_0
j	xp	$n-x$	i_0

• Diagram of vector dimensions.



Y by Z, and Z again by X. Both record areas and control words may be scattered throughout memory. Note that in this notation capital letters are used for record areas and lower case letters for control-word locations. Corresponding letters are used in each control word to denote a record area and the control word of the *next* area in sequence.

The example of Fig. 5 shows the sequence of operations in a read-process-write cycle. While a record is read into Area Z, as controlled by control word $Z-z$,

Table 5 **Program for matrix multiplication.**

• Instructions

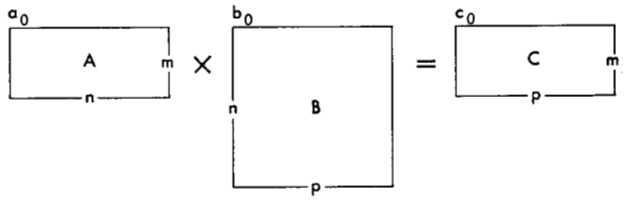
Preparation	$h-2$	Load k from k_0
	$h-1$	Load j_0 from j_{00}
Initial setup	h	Load i_0 from i_{00}
New product row procedure	$h+1$	Load i from i_0
New vector product procedure	$h+2$	Load j from j_0
	$h+3$	Set accumulator to zero
Vector multiply, inner loop	$h+4$	Load cumulative multiplicand from location specified by j
	$h+5$	Multiply cumulatively by operand location specified by i
Housekeeping, inner loop	$h+6$	Increment j by p
	$h+7$	Advance i, count, refill when count reaches zero, branch to $h+4$ when count does not reach zero
End of vector multiplication procedure	$h+8$	Store cumulative product at location specified by k
	$h+9$	Increment k by 1, count, refill when count reaches zero
	$h+10$	Advance j_0, count, refill when count reaches zero and branch to $h+2$ when count does not reach zero
End of product row procedure	$h+11$	Increment i_0 by n
	$h+12$	Compare index values of k and k_0
	$h+13$	Branch to $h+1$ if comparison result was not equal

• Control Words

Contents after executing the inner loop x times for the product matrix element c_{rs}

Address	*Index Value*	*Count*	*Refill*
i	a_0+rn+x	$n-x$	i_0
i_0	a_0+rn	n	i_0
i_{00}	a_0	n	i_0
j	b_0+s+xp	$p-s$	j_{00}
j_0	b_0+s	$p-s$	j_{00}
j_{00}	b_0	p	j_{00}
k	c_0+rp+s	$mp-rp-s$	k_0
k_0	c_0	mp	k_0

• Diagram of matrix dimensions.



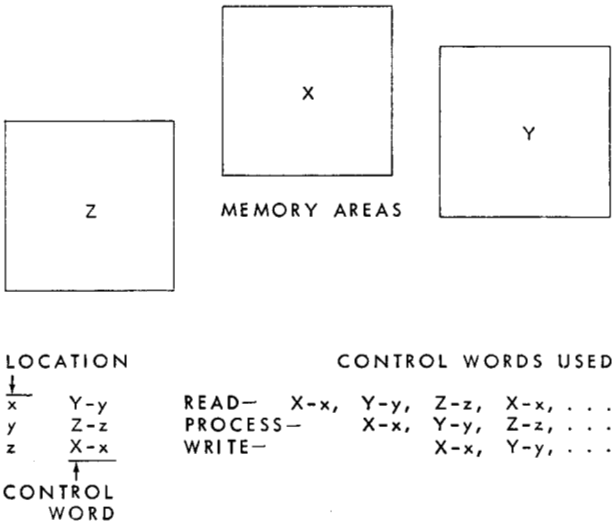


Figure 5 **Read-process-write chain.**

processing proceeds with control word $Y-y$ using data in Area Y, and data from Area X are written under control of control word $X-x$. At the conclusion of each of these operations, the appropriate control word is refilled and the areas are thereby cyclically permuted in function.

Instead of a single control word, a chain of n control words could be used in reading while a second chain of n control words is used in processing, and a third chain of n control words is used in writing. To further elaborate the example, assume that processing consists in placing the n records in a preferred sequence. This sequencing operation was described above. Because of the refill field, however, the control words do not have to be in sequential locations. The advantage of this added degree of freedom will be shown in the following examples.

Assume that the records $A \ldots Z$ are scattered throughout memory. The associated control words $A-a \ldots Z-z$ establish their order. The correct order is indicated by the alphabetic sequence. It is desired to delete Record H, which is out of sequence and set its memory area aside. The control word $H-h$ of this record is part of the chain $C-c \ldots K-k$ shown in the left half of Table 6. By interchanging the contents of locations d and h, a new order is established as shown in the right half of Table 6, and H is no longer part of the sequence. A second interchange between d and h would re-insert H. Thus, the complementary nature of insertion and deletion is reflected in the programming procedure.

If it would be desired to insert H in the sequence $\ldots G, I, J, \ldots$ between G and I, the second interchange would be between g and h. Table 7 illustrates this case.

Because the sequence $\ldots G, I, J \ldots$ is part of sequence $A \ldots Z$, the example is equivalent to a sorting operation. The sequence $\ldots G, I, J \ldots$ may equally well be part of an independent sequence, as is the case in file maintenance.

Table 6 **Record deletion.**

• Before		• After			
Location	*Control Word*	*Location*	*Control Word*	*Location*	*Control Word*
....	....	....	....		
b	*C—c*	*b*	*C—c*		
c	*D—d*	*c*	*D—d*		
d	**H—h**	*d*	**E—e**		
h	**E—e**			*h*	**H—h**
e	*F—f*	*e*	*F—f*		
f	*G—g*	*f*	*G—g*		
g	*I—i*	*g*	*I—i*		
i	*J—j*	*i*	*J—j*		
j	*K—k*	*j*	*K—k*		
....	....	....	....		

Table 7 **Record insertion.**

• Before				• After	
Location	*Control Word*	*Location*	*Control Word*	*Location*	*Control Word*
....	....			....	....
b	*C—c*			*b*	*C—c*
c	*D—d*			*c*	*D—d*
d	*E—e*			*d*	*E—e*
e	*F—f*			*e*	*F—f*
f	*G—g*			*f*	*G—g*
g	**I—i**			*g*	**H—h**
		h	**H—h**	*h*	**I—i**
i	*J—j*			*i*	*J—j*
j	*K—k*			*j*	*K—k*
....	....			....	....

Table 8 **Group deletion.**

• Before		• After			
Location	*Control Word*	*Location*	*Control Word*	*Location*	*Control Word*
....	....	....	....		
b	*C—c*	*b*	*C—c*		
c	**P—p**	*c*	**D—d**		
p	*Q—q*			*p*	*Q—q*
q	*R—r*			*q*	*R—r*
r	**D—d**			*r*	**P—p**
d	*E—e*	*d*	*E—e*		
e	*F—f*	*e*	*F—f*		
f	*G—g*	*f*	*G—g*		
g	*H—h*	*g*	*H—h*		
....	....	....	....		

The interchange of two control words is performed conveniently by a SWAP instruction. This instruction interchanges the contents of two memory words. The insertion or deletion of a record involves only the SWAP of its control word with that of its successor. The insertion and deletion of a group of records is equally simple. Consider again the file $A \ldots Z$. It is required to delete the group $P \ldots R$ from the file shown on the left in Table 8. By giving a SWAP instruction for locations c and r, the new order becomes as shown on the right in Table 8.

One SWAP instruction deletes the group of records just as one SWAP instruction in the previous example deleted a single record. The only differences are the addresses of the instruction. The records $P \ldots R$ form a ring in sequence. (In the previous example, the deleted record H could be considered to form a ring in sequence, since its control word was stored at its own refill location.) The re-insertion of the records $P \ldots R$ can be performed by again swapping the contents of locations c and r.

In these examples the sequence of control words is changed by transmitting entire words. A different approach is to transmit refill fields only, leaving the remainder of the control word unchanged in memory. This method can also be used in many applications.

File maintenance

A simple case of updating a master file from a detail file will be discussed. Four tapes are used: the old master tape, the new master tape, the detail input tape, and the detail output tape. The detail records are processed in a simple input-process-output operation as was described above. The master records are ready from the old master tape, processed and written on the new master tape. Reading, writing and processing take place simultaneously. The processing of a master record may involve:

a) no activity,
b) updating,
c) deletion of obsolete records, and
d) insertion of new records.

Master records are read and written in groups of m records. Memory space is set aside for a total of $4m$ master records and their control words. Normally, m records are written on the new master file, while m records are read from the old master file. The remaining $2m$ record spaces are available for processing. These record spaces are divided into two groups: the *current* spaces and the *spare* spaces. The current record spaces contain records which either have been processed and are ready to be written on the new master tape, or which have been read from the old master tape and are available for processing. The spare record spaces contain no useful record information. The number of current and spare spaces varies throughout the processing but their sum remains $2m$.

The control words used in reading and writing and the control words of the current records form a ring. The control words for the spare record areas also form a ring. Figure 6 shows the control words in diagram form and illustrates the cases discussed below for $m=8$.

When a record is inactive or requires updating, the number of current and spare records remains unchanged. The record is addressed by means of its control word. After the processing is completed, the current control word is replaced by the next one in order by means of a REFILL instruction. The record is ready to be written on the new master tape. A count is kept of the records which are ready to be written. When the count equals m, a WRITE instruction is issued which is followed by a READ instruction. The record space of the records just written is used for the records to be read. The records just read are available for processing.

When a record is found to be obsolete and should be deleted, its control word is removed from the ring of current control words and inserted in the ring of spare control words. Because the control word is deleted, its record is not written on the new master tape. The count of records which are ready to be written is not changed. The control word of the next record is obtained and processing continues.

Because of an excess of deletions, all current records may be processed before m records are ready to be written. In that case, the number of spare record areas is always larger than m and a corrective step can be taken. This step consists in deleting m control words from the spare ring and inserting them in the read-process-write ring. The control words are inserted as a block preceding the control words used in reading and following those used in writing. An extra READ instruction is given and processing proceeds with the records which have just been read.

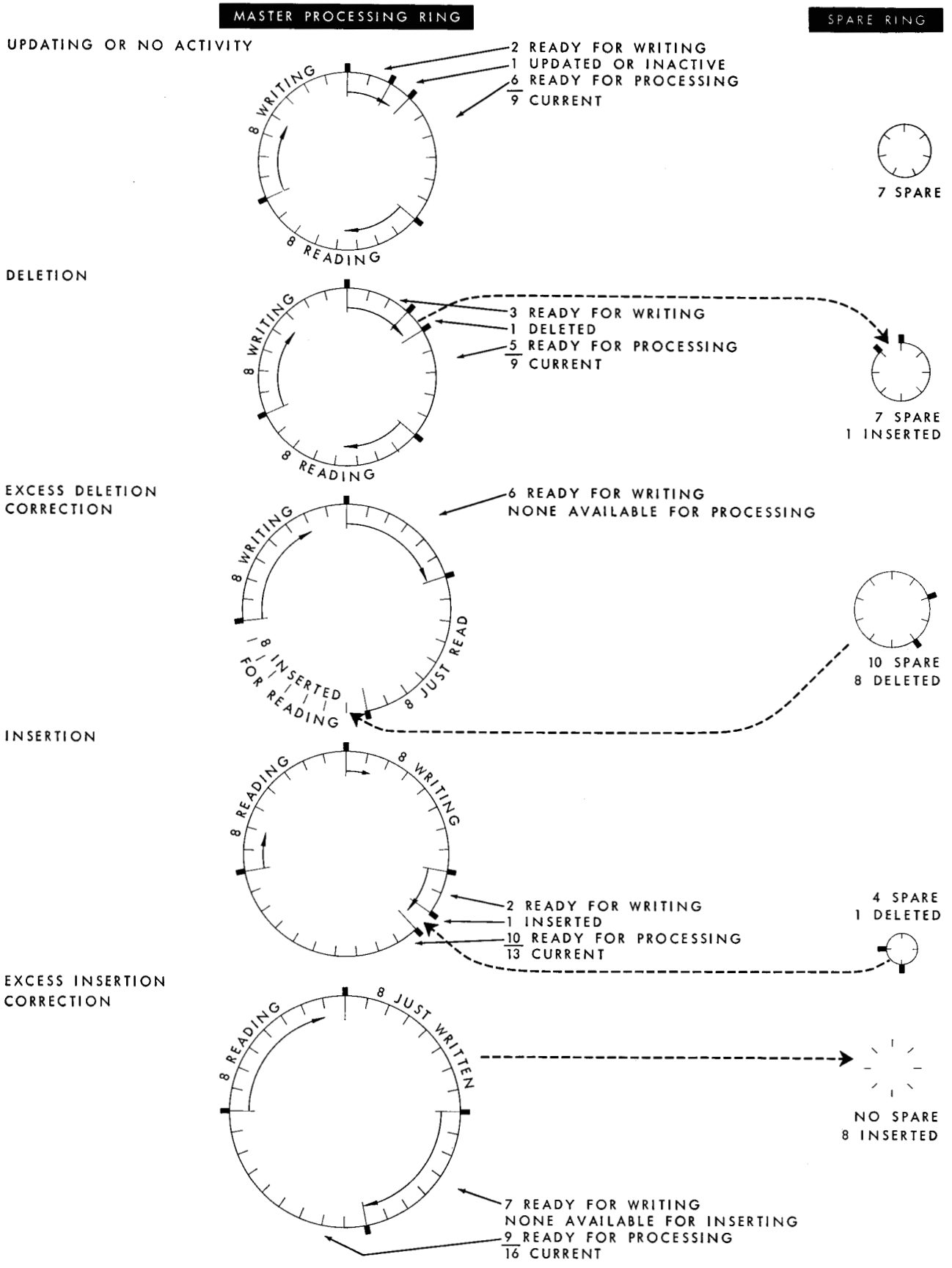
When a new record is to be inserted, a control word is removed from the ring of spare control words and inserted in the ring of current control words. The corresponding record area is then available for the new record. After the new record is processed, it is ready to be written.

Because of an excess of insertions, the spare control-word ring may have been reduced to zero. A corrective step then should be taken by deleting m control words from the read-process-write ring and using them as a new spare ring. The m control words which are deleted are those which were last used in a write operation. Writing is checked for completion. The next time that m records are ready to be written, the WRITE instruction is given but the READ instruction is omitted.

The file maintenance procedure outlined above illustrates the use of insertion and deletion of single records and groups of records. All the manipulations which were described are performed conveniently with control words and would require a great deal of housekeeping without the refill feature.

Subroutine control

Another application of control words is in subroutine control. In the preceding discussion, the control word



300 **Control-word diagram for file maintenance.**

specified a memory area which normally would contain data. However, the memory area might also contain instructions. A record can then be thought of as a subroutine. An illustration might be the use of exception subroutines which are stored on tape, drum or disk, and are called in when the exception arises. The control word is used in the READ instruction and can subsequently be used for address modification in the BRANCH instruction which refers to the subroutine and in the instruction which stores the instruction counter contents. The subroutines, therefore, can be inserted conveniently in a main sequence of instructions.

The chaining concept has been developed independently by Newell, Shaw and Simon, who have shown many interesting examples of its function on a simulated computer.[4]

Conclusion

The preceding discussion has shown the application of control words in address modification and in record handling. Both indexing and data transmission techniques make it desirable to have an index value, count and refill facility. The three fields in the control word and the associated machine functions satisfy these requirements. The control words provide substantial saving in program space and increase in machine speed. They simplify programming of "housekeeping" operations.

Control words do not introduce entirely new functions, since their operation can be simulated on any stored-program computer. Also, the introduction of count and refill is only a second-order improvement as compared to the first-order improvement of address modification through indexing. The simplicity of control-word operation is, however, in sufficient contrast to the complexity of simulating its operation that several methods of record control are feasible which otherwise would have been impractical.

The indexing instructions have been described for the STRETCH Computer. Though elements of the system described here have been used in other machines, the effectiveness of control-word techniques depends to a major extent upon the combination of all features which have been described. It is believed that control-word techniques represent a significant step forward in the data-handling ability of computers.

Acknowledgments

Much of the early development of the control-word concept was stimulated by discussions with G. M. Amdahl, E. M. Boehm, J. E. Griffith and R. A. Rahenkamp. Many contributions to the use of control words in the STRETCH Computer were made by W. Buchholz, F. P. Brooks, Jr., and C. A. Scalzi.

References

1. T. Kilburn, "The University of Manchester High-Speed Digital Computing Machine," *Nature*, **164,** 684 (1949).
2. W. L. van der Poel, *The Logical Principles of Some Simple Computers,* Excelsior, The Hague, Netherlands, p. 100.
3. The following technical papers have been published about the STRETCH computer:
   a) S. W. Dunwell, "Design Objectives for the IBM Stretch Computer," *Proceedings of EJCC,* p. 20 (December, 1956).
   b) F. P. Brooks, Jr., "A Program-Controlled Program Interruption System," *Proceedings of the EJCC,* pp. 128-132 (December, 1957).
   c) W. Buchholz, "The Selection of an Instruction Language," *Proceedings of the WJCC,* p. 128 (May, 1958).
   d) F. P. Brooks, Jr., G. A. Blaauw, W. Buchholz, "Processing Data in Bits and Pieces," *IRE Transactions on Electronic Computers* (June, 1959).
4. a) A. Newell, J. C. Shaw, "Programming the Logic Theory Machine," *Proceedings of the WJCC,* p. 230 (February 1957).
   b) A. Newell, J. C. Shaw, H. A. Simon, "Empirical Explorations of the Logic Theory Machine," *Proceedings of the WJCC,* p. 218 (February 1957).
   c) A. Newell, H. A. Simon, "The Logic Theory Machine," *Transactions on Information Theory,* **IT-2,** No. 3, 61 (September 1956).
   d) J. C. Shaw, A. Newell, H. A. Simon, T. O. Ellis, "A Command Structure for Complex Information Processing," *Proceedings of the WJCC,* p. 119 (May 1958).

Received August 22, 1958