R V SMITH

June 27, 1956

Memorandum for: File

Subject: Indexing for the Improved 705

Indexing is a procedure whereby the memory address of an instruction is modified by the contents of an index register before it is used. Usually the contents of the index register are added to or subtracted from the memory address. The number contained in the index register is called the index. Since there is usually more than one index register, several indices are available for use. The index selected is shown in a specific place in the instruction and represents the value to be used in conjunction with the memory address of the instruction to determine the effective memory address. This is not to be confused with the index addressed; in this case an index register is addressed for the purpose of taking data from it or putting data in it.

In a strictly one-address machine, the index addressed would be shown in the instruction where the memory address would otherwise be. However, greater flexibility may be obtained by introducing the index addressed into the instruction as a second address. For reasons which will appear later, the index addressed and the index selected should be shown in two different and independent locations in the instruction. In the proposed improvement for the 705, the index selected and the index addressed are indicated in the lead word as shown in figure 1; in addition, there is a use-index bit in the operation word which shows by its presence that the index selected is to be used with that operation word.

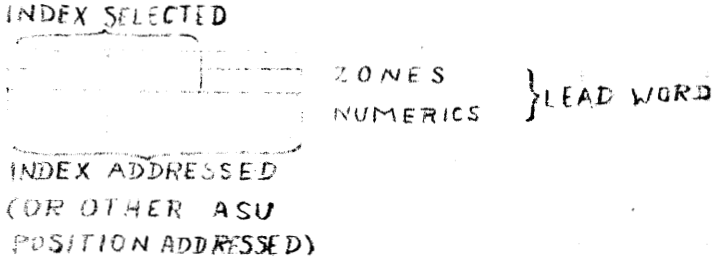

Figure 1

The value of indexing is found in relative programing and in loop programing. In relative programing, the characters of a data area or work area are assigned relative addresses, the first character being given the address 00000 and each succeeding character receiving an address one number higher. An index is selected and used with each instruction addressing the area; by setting the index to the actual address of the first character of the data area, the effective address of the instructions involved will be properly set. This technique is best used when either the location of the data area is unknown and must be developed during the program and/or when there are a number of similar data areas to which the instructions will apply. Only one number, the index value, must be changed to change the effective address of all instructions for which this index has been selected.

Indexing is an aid to loop programing prinicipally when there are a number of instructions to be applied successively to several work areas. For example, the entire processing of grouped records must be repeated for each record of the group. Such a program may be written using the actual addresses of the first record. Each instruction of the program would select and use an index whose value would be set to zero for the first record and increased by (say) 200 for each successive 200 - character record. The end of the loop is reached when the index attains a certain maximum value, which depends on the particular case. If 4 records are involved, the last record would be processed with an index value of 3 x 200 = 600. Note that only one number, the index, has to be changed when stepping from one record to the next. If relative programing is combined with looping, the index starts with the actual address of the first character of the first record and is stepped up the amount of the record length, say 200, until the last record is processed with an index value equal to the actual address of its first character.

To obtain the maximum value from indexing, it should be possible to index any program. It is quite possible that a program which we wish to index through a series of grouped records will itself contain indexing. If the grouped records are being handled with simultaneous read-write-compute, it will be desirable to index the program through the several compute areas as well as through the records of the group. Thus a third (or even higher) level of indexing may be required. This multiple indexing should be possible in any indexing scheme.

The elements of looping are shown in figure 2.

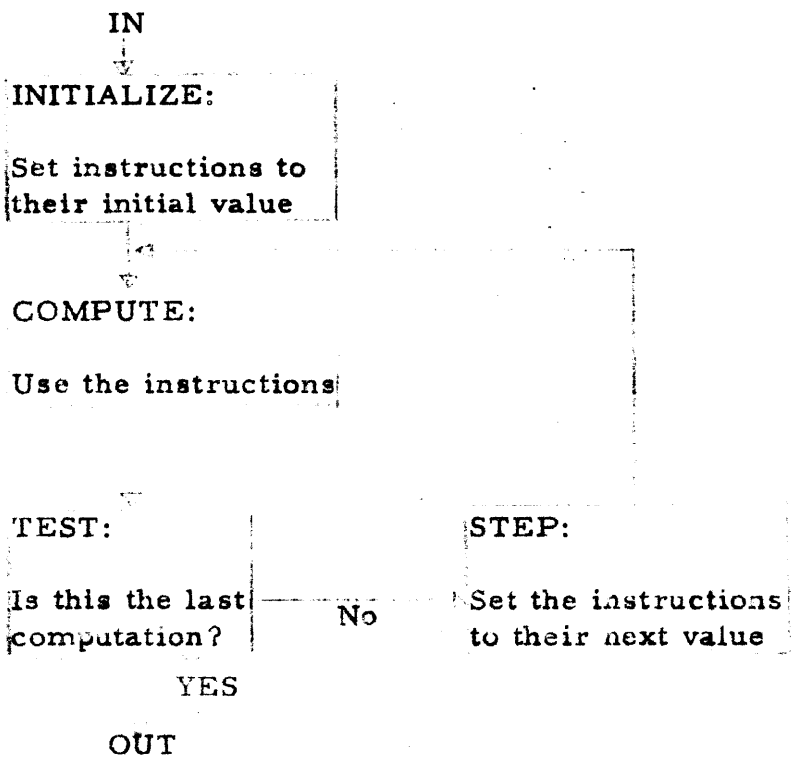

Figure 2

With indexing, looping becomes a matter of initializing, using, testing, and stepping an index. The problem of establishing an indexing scheme comes down to a consideration of these four operations.

One possible plan for indexing is to store in memory the initial value, the maximum value and the stepping value for the index. In initializing, the initial value is loaded into the index register; in testing, the maximum value is compared against the index value; in stepping, the stepping value is added to the index. A more flexible plan is to place these constants in the memory address of the initializing, stepping and testing instructions. The operations required for this plan are:

Phase:	Nature of Instruction:
Initializing	Load (reset-add) memory address into index
Stepping	Add memory address to index
Testing	Compare memory address to index

The advantages of this plan are: (1) no memory space is required for the constants other than that required for the instructions - this also simplifies the programmer's job; (2) since no memory access for the constant is involved, time may be saved; (3) the constants may be modified by indexing the instructions containing them - this is the key to multiple indexing.

In one type of indexing, (available on the 704 through the TIX instruction) the index steps from an initial value, set by the problem, down to zero. The indexing system proposed in 705 Improvement Memo #3 is similar to this except that the index is stepped upward until its value reaches zero through an overflow. Both schemes have the same limitation: the initial and final values of the index are fixed so that the memory address plus the index (the effective memory address) has a fixed initial value and final value. This allows only one starting place and one ending place as, for example, starting with the first record of a group and indexing through the last record of the group. A second-layer index to permit directing the program successively to different compute areas is not possible with this type of indexing.

The proposed improvement plan for the 705 provides operations with indexing constants stored in the memory address as follows:

LFM - Load from memory address - loads the memory address into the index addressed

MODX - modify index - loads the memory address into the index selected

Either of these instructions can be used to initialize an index. The MODX instruction violates a basic concept inasmuch as the index selected is taken to be the index addressed; however, no E-time is required so that the MODX instruction takes less time than the LFM instruction. If in the MODX instruction, the use-index bit is present, self-indexing takes place: the memory address is indexed by (i.e., added to) the index selected and the result loaded into the same index. This is equivalent to stepping the index by the amount of the memory address. The same effect can be obtained with the LFM instruction by indicating the same index as index selected and index addressed. When the index-used bit is present, the index will be stepped by the amount of the memory address. When the LFM instruction is used to initialize an index, its memory address may be indexed by another index - - this technique is used in multiple indexing. The MODX instructions cannot be used in this application. Neither the MODX or the LFM instruction can be indexed with another index when they are used for stepping because the stepping is done by self-indexing; no application for such indexing has as yet turned up and so there is perhaps no valid objection to the lack of a direct add-from-memory-address instruction.

Attached to this memorandum as figure 3 is a sample program written for multiple indexing using the MODX and LFM instructions. The program is written for grouped records, 200 characters long, with a header record of 100 character in front of each group. The second-layer index is used to progress the program through three compute areas as is commonly done with simultaneous reading-write-compute. Since the proposed 705 improvement does not include an instruction for the testing phase, an instruction marked "TEST" has been added with the assumption that this will compare the memory address (after indexing if so indicated) to the index selected and set the equal trigger on or off in accordance with the result of this comparison. (This assumption is for illustrative purposes only; for instance, a special index-equal trigger might be preferable.)

The basic requirements of a complete indexing system are: (1) an instruction format that indicates separately and independently the index selected and the index addressed; (2) an instruction which loads the memory address into the index addressed - by self-indexing this will also add the memory address to the index addressed; (3) an instruction which compares the memory address with the index addressed and sets a trigger accordingly. These features are strongly recommended for incorporation into the indexing plan for the improved 705.

RVS:pw
Attachment

R. V. Smith

Multiple Index Program - Improved 705

Class	Symbolic Location	Oper.	Index Or ASU Address	Memory Address	Use. Ind.	Index Sel.	Description
7							Initialize
1	00010	LFM*	009	00000			00000 to IR 009
1	00020	LFM	004	00000	+	009	00000 + IR009 to IR004
7							Calculate Routine
	10010	000	AAA	MMMMM	+	004	MMMMM etc are actual addresses for first record
	10020	etc.	etc.	etc.	+	004	in first area
7							Test End Minor Loop
1	20010	TEST	004	00600	+	009	Last (4th) record in group?
	20020	TRE		40010			Yes
7							Step Through Minor Loop
1	30010	MODX		00200	+	004	No-step to next record
	30020	TR		10010			repeat calculation routine
7							Test End of Major Loop
1	40010	TEST	009	01800			Last (3rd) area?
	40020	TRE		out #1			Yes - re-enter at 00010 after RD
7							Step Through Major Loop
1	50010	MODX		00900	+	009	No-step to next area
	50020	TR		Out #2			re-enter at00020 after RD

* If there is no higher layering of indices, instruction 00010 should be written:

1	00010	MODX	00000	009

Figure 3