A Lemma for Automatic Optimum Programming on the IBM 650

1. Introduction. The important problem of optimum programming a set of instructions for the IBM 650 Magnetic Drum Data Processing Machine has been attacked in several ways [1, 2, 3, 4]. The techniques commonly used, whether they be manual or machine operated, refer to the optimum programming table for the minimum spacing of instructions and data on the drum [5]. Perhaps the major limitation in current procedures is that the program becomes less nearly optimum as succeeding instructions are placed; i.e., unless the program contains only relatively few instructions that need not be densely located on the drum, once half of the program has been placed optimally the second half is located on a "catch-as-catch-can" basis and reference to the optimum programming table for placement becomes virtually useless.

It is the purpose of this paper to present a lemma which has proved useful for automatic optimization of 650 routines.

2. Fundamentals of 650 Coding. The main storage medium of the 650 is a revolving (12,500 r.p.m.) magnetic drum containing 1000 or 2000 locations, which hold signed 10-digit words of data and instructions. Drum locations, assigned addresses 0000, 0001, $\cdots$, 1999 (assuming a 2000 word drum), are arranged in sequential order, starting at 0000, in strips of 50 words around the surface of the drum. In addition to drum storage, there are four other special locations, addressable as 8000, 8001, 8002, and 8003.

A 650 instruction is composed of three parts, the two most significant digits specifying the operation, the next four digits indicating the Data Address, and the last four digits indicating the next Instruction Address (the sign of an instruction has no effect on the operation). The Data Address usually denotes the location for a) obtaining an operand, b) storing a result, or c) reading or punching information; the Data Address may also specify the length of a shift or the address of the next instruction that follows a logical test in which the test condition (e.g., non-zero accumulator) has been satisfied. The Instruction Address usually refers to the location of the next instruction, the exception being after a logical test. Both the Data and Instruction Addresses may be any drum location or 800X, with the qualification that 800X may not be used as a Data Address in input-output, store, or table look-up commands.

The presence of a next Instruction Address allows one to disperse instructions as well as data around the drum; hence for the most part, coding a sequence of required operations may be undertaken without regard to the ultimate placement of instructions and data. Storage is finally assigned to reduce waiting time as much as possible, i.e., the time between the moment the computer is ready to receive some data or an instruction and the moment the data or instruction arrives under the drum read-write heads. Because the sequence of operations and the assignment of storage are practically distinct problems, minimum latency coding in the 650 is conceptually easier to handle than, say, in mercury delay line systems, where instructions are placed in successive locations, and, as a result, the sequence and assignment questions become inseparable.

3. The Optimizing Lemma. Let n and k be positive integers. Then the sequence

a) $1, 1 + n, 1 + 2n, 1 + 3n, \cdots$ modulo $(nk + 1)$ completely exhausts the numbers $(1, 2, 3, \cdots, nk + 1)$ before repeating any number in the sequence; similarly for

b) $1,\ 1 + n,\ 1 + 2n,\ 1 + 3n,\ \cdots$ modulo $(n(2k + 1) + 2)/2$ where n is even; and

c) $1, 1 + n, 1 + 2n, 1 + 3n, \cdots$ modulo $(n(2k + 1) + 1)/2$ where n is odd.

For example, let $nk + 1 = 10$ and $n = k = 3$; then we have the sequence 1, 4, 7, 10, 3, 6, 9, 2, 5, 8, which completely exhausts the numbers 1 through 10 without any repetition.

If, in an effort to reduce access time on the 650, we desire to place a sequence of N consecutive instructions and data (a precise definition of "consecutive" is given below) n locations apart, we find an integer k such that one of the cases in our lemma applies. Thus we may place a set of 50 consecutive instructions and data 7 locations apart on a band of 50 locations: 0000, 0007, 0014, $\cdots$, 0049, 0006, $\cdots$, 0043 ($n = k = 7$, case a) of the lemma). Furthermore we may place on the drum any set of N consecutive instructions and data by first densely filling up r^* bands of 50 locations, where r^* is the largest integral value of r such that $N - 50r \geq 0$. The remaining $N - r^*50$ instructions are placed according to the lemma *within* the next block of $7k^* + 1$ successive locations, where k^* is the smallest integral value of k such that $7k + 1 \geq N - 50r^*$. If storage space is not limited a better choice is $k^* = 7$. As the reader can see, the previous example generalizes to any N and n and need not utilize integral multiples of 50 locations.

4. Sequential (or Serial) Coding. We shall now clarify the notion of a "sequence of consecutive instructions and data." Consider a given 650 instruction stored in location j. If the instruction's operation is add, subtract, multiply, divide, store, or load, we shall code the Data Address as location $j + 1$ and the next Instruction Address as location $j + 2$. If the Data Address is 800X, we may code the Instruction Address as $j + 1$, or as $j + 2$ and void the location $j + 1$; of course the Instruction Address may be 800X if desired. On branching operations we should let location $j + 1$ be the most traversed route; the address remaining will refer to some other part of the program. In shift operations we code the Instruction Address as location $j + 1$. Most routines will not have all their instructions and data in strictly consecutive order, since branching operations may be included and since some data locations may be referred to more than once in a program. The optimizing routine suggested here operates best if the program is originally written as nearly sequentially as possible.

The coder observes the following procedure in writing a new routine: He codes his program in as nearly a "sequential order" as is practicable with real drum addresses starting, say, at s_t and ending at s_e, and has it punched on any standard one/card form. *He next debugs the sequential code on the 650.* As experienced 650 programmers will attest, the sequential pattern greatly aids the checking out procedure. Any additional operations which must be included are added in locations immediately preceding s_t or following s_e. The final debugged program is then located between, say, s_f and s_g; note that the program need not be dense between s_f and s_g. The one/card form contains in addition to the location of the instruction

and the instruction itself, coded information analogous to that needed for automatic translating routines, viz., a number XX (99, 98, 89, or 88) referring to whether or not the Data or Instruction Addresses are translatable (9 being not-translatable).

5. Automatic Optimizing. To program the debugged routine, the coder inserts in the optimal programming deck an "information card" which has punched in it s_f, s_g, and o_f (the first location for the optimal program). He places the one/card debugged routine behind the optimizing deck and runs the combined deck through the 650; a new one/card optimized deck is punched out.

The optimizing routine actually consists of two separate parts.

We shall assume that some n and some modulus have been selected to satisfy a particular case in our lemma. In the first part, given the addresses s_f and s_g, the 650 computes how many locations are needed for the optimized routine. Starting with address o_f, the machine constructs the sequence of addresses as prescribed by our lemma, and concomitantly, the addresses in the new sequence are associated with and correspondingly stored in locations s_f through s_g. When the first part of the program has been completed, the second part commences by reading each card in the one/card debugged deck, optimizing its contents, and subsequently punching the new optimized one/card instruction. From the 9's and 8's information punched in the original card, each instruction is altered so as to replace where indicated a Data or Instruction Address in the range s_f to s_g with its associated new optimal address. The correct new location of each instruction in the sequential deck is determined by testing whether the old instruction location lies within the range s_f to s_g; if it does not, the location is not altered. Hence cards used in the sequential program deck with instructions and data to be placed in absolute (i.e., fixed or unalterable) addresses may remain in the deck which is being optimized; although the fixed address will not be altered, necessary Data and Instruction Address changes are made as indicated by the 9's and 8's coding.

We may now summarize the programming limitations imposed by this method of optimizing: No absolute address may lie either within the range s_f to s_g or o_f to o_g (the last optimized location). Depending on the particular optimizing program devised, the range s_f to s_g may or may not be allowed to overlap with the locations in which the optimizing program is stored. There are no other restrictions on s_f, s_g, o_f, o_g, other than that they must be legitimate 650 addresses, and s_f to s_g may overlap o_f to o_g.

6. General Comments. It should be obvious to the reader that the proposed scheme has the property that the degree of optimization is essentially invariant to the number of instructions previously optimally located. Thus in a program utilizing 500 locations, the instructions and data can be placed optimally and densely in 500 consecutive locations, and the degree of optimization for the last 100 instructions and data is essentially the same as that for the previous 400 instructions and data. It is the author's belief that this property will prove to yield results at least as good as those obtained by "strict" optimization of the instructions in the first half of a program and "catch-as-catch-can" allocation of the instructions in the second half.

Using D. W. Sweeney's statistics [6] for the average use of 650 commands,

we may expect our lemma for $n = k = 7$ (case a)) to be about three times as fast as sequential coding. (In one program coded by the author, the optimizing of a payroll calculation resulted in a reduction in computing time sufficient to cause calculations to take place at full reading and punching speeds.) A desirable property of the suggested optimizing code is that a single routine may be written for one case of the lemma and arbitrary n and modulus; assignment of particular values to these parameters may be done by the insertion of a single card in the optimizing deck. Therefore a sequential program may easily be optimized for several different values of n and modulus in order to test the best mode of optimizing for the particular routine. If a program contains mostly add, subtract, store, and branch commands, computing speed may be increased by using $n < 7$. Given n, one should select a case of the lemma and k to maximize (modulus) ≤ 50.

Although the lemma is simple enough to use as a manual technique for concomitantly coding and optimizing, we do not recommend the procedure, because testing out a sequential routine has important time saving advantages and because the automatic method for optimizing is so easy and fast.

A routine using case a) of the lemma has been written by the author for M.I.T.'s Statistical Services; the program occupies less than 100 locations and will optimize a one/card deck at full punching speed of 100 cards/minute.

HARVEY M. WAGNER

Massachusetts Institute of Technology
Cambridge, Massachusetts

1. B. GORDON & A. J. DALTON, "Optimizing program," Equitable Life Assurance Society and IBM, New York, 1955 (mimeographed).
2. BARRY GORDON, "An optimizing program for the IBM 650," *J. Assn. for Comp. Machinery*, v. 3, 1956, p. 3–5.
3. S. POLEY & G. L. MITCHELL, "S. O. A. P., IBM 650 symbolic optimal assembly program, programmer's guide," IBM, New York, 1955.
4. E. F. SHEPHERD, "An automatic method of optimum programming," *IBM Technical Newsletter No. 10*, New York, 1955, p. 95–104.
5. IBM, "Type 650 manual of operation, first revision," New York, 1955, p. 70–89.
6. D. W. SWEENEY, "A note on optimum programming and the IBM type 650 operation code usage," *IBM Technical Newsletter No. 10*, New York, 1955, p. 105–107.

REVIEWS AND DESCRIPTIONS OF TABLES AND BOOKS

49.—A. V. LEBEDEV & R. M. FEDOROVA, *Spravochnik po Matematicheskim Tablitsam* (*Guide to Mathematical Tables*), Moscow, 1956, xlvi + 549 p., 26 cm. Price 29.20 rubles.

In this *Guide to Mathematical Tables* the authors have extended the largest available index to mathematical tables, Fletcher, Miller, and Rosenhead [1], to include tables published in books up through 1952 or later and tables published in journals through 1953. These dates seem more recent than the closing dates for Schütte's index [2].

The preparation of this index was necessitated by the great activity in table making and related parts of numerical analysis in USSR. The *Guide* was prepared as a convenience both to scholars and to practical workers. Thus, it is to serve a somewhat wider class of users than Fletcher, Miller, and Rosenhead [1] or