THE LOGICAL DESIGN OF A SERIAL GENERAL PURPOSE DIGITAL COMPUTER WITH MICRO-PROGRAM CAPABILITIES

Thesis by

Richard Isamu Tanaka

In Partial Fulfillment of the Requirements

For the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

1958

ACKNOWLEDGEMENTS

The author wishes to express his appreciation to Professor C. D. McCann for his kind cooperation, aid, and interest in the undertaking. Thanks are due also to Professor R. V. Langmuir for his many helpful and encouraging comments.

The manuscript was prepared by Miss Edith Hamilton, assisted by Miss Esther Samit of the Hughes Aircraft Company, who prepared the micro-routine tables. The figures were drawn mainly by Miss Phyllis Klein of Hughes Aircraft Company. The author gratefully acknowledges their patient and invaluable assistance.

The author wishes also to acknowledge the encouragement, support and understanding of his wife, Edith, whose help made possible this thesis.

Portions of the research were conducted while the author was the recipient of a Howard Hughes Fellowship in Science and Engineering; the author is deeply indebted to the donors of this Fellowship.

ABSTRACT

This thesis presents the detailed logical design of a serial, general-purpose digital computer with micro-program capabilities. The machine is a medium-speed computer with a magnetic drum memory, and includes a modest magnetic core memory for storing micro-orders.

The micro-program feature relates to the command structure of the machine. Micro-orders enable the programmer to specify many of the elementary internal operations; these micro-orders are arranged into micro-routines which synthesize computation sequences equivalent to commands in more conventional computers. Therefore, effectively, the machine has an extremely long command list.

Included are descriptions of the internal organization of the machine, and the detailed logical equations which are directly applicable to the construction of the computer. A few micro-routines, for both usual and uncommon commands, are also shown, to illustrate possible applications of the machine.

TABLE OF CONTENTS

I INTRODUCTION

Early in the development of electronic digital computers, when successfully operating machines were so few in number that each was known by name, each existing machine was operated almost exclusively by programmers who were thoroughly familiar with the machine language and internal characteristics of their particular computer. (In many instances, the programmers themselves had been closely associated with the original design of the machine.)

However, with the increasing availability of digital computers, the expansion in machine capabilities, and the overall improvements in reliability, it became increasingly difficult to supply an adequate number of experienced programmers to staff the machines. In an effort to minimize this need, the coding effort was transferred, in many instances, to those responsible for the problems. As a result, the casual user, whose sole objective is the solution of his particular problem, has become commonplace, and there now exists direct contact between computers and a large segment of technical society which has no direct interest in knowing how the machines function internally.

Both effects, the limitation in experienced-programmer time and the blossoming of the occasional user, have emphasized the necessity for developing a language more convenient for communicating with computers. A major development in this direction is in the growth of automatic coding techniques. The use of assembly and compiling routines or of interpretive routines, though less efficient in the use of

internal machine time, nonetheless usually results in an overall decrease in the time required for problem solution. A greater share of the burden is shifted to the computer, removing much of the drudgery from the programmer's hands, and often eliminating most of the need for a detailed knowledge of the computer's internal operation.

In some instances, in fact, automatic coding techniques have been used to describe "pseudo-computers" (1), whose characteristics may appear, to the user, very different from those of the actual machine being used (hopefully, simplifying the problem for the user).

But automatic coding techniques are usually applied to large-scale digital computers which share many of the characteristics of high internal speed, large random-access storage, extensive input-output capabilities, and supplementary high capacity storage.

For the smaller computer, where compromises have been made in favor of cost, size, power requirements, etc., assembly and compiling techniques or interpretive routines are less practical to apply. The more restricted command list often causes the routines to be longer, and these are then executed at the inherently lower operating speeds.

This thesis proposes a method for increasing the versatility of smaller digital computers by changing the fundamental command structure now shared almost universally by existing digital computers. The usual command list is expended, not by cascading further orders into the machine, but by allowing the programmer to exercise a greater degree of control over the internal process of the computer.

The proposed scheme is described as "micro-programming". The discussion in Part II indicates that the "micro-programming" referred to in

this thesis differs markedly from that mentioned in references available in the literature, but the name is adopted since it seems best to describe the process used.

The use of micro-programming techniques does not entirely eliminate the need for a detailed knowledge of the internal characteristics of the computer. In many instances, the development of the micro-routines which make up the command list is very similar to the process used in the logical design of more conventional computers. However, it is anticipated that libraries of micro-programs would be made available to the casual user, so that he would have the choice of a very long command list, utilized in more or less standard fashion. The command list could well include commands which, though convenient, are usually not built into machines because of their infrequent use. It would appear to be feasible to duplicate the command list of other machines, perhaps, and utilize routines already in existence. The more experienced programmer would hopefully be able to operate the computer in more sophisticated modes.

The following, Part II, contains an introduction to the existing literature on micro-programming and defines various terms used throughout this thesis. Part III describes the general characteristics of the computer and its internal command structure; a detailed description of the logical design follows in Part IV.

Sample micro-routines are discussed in Part V, and illustrate the capabilities and limitations of the machine. Part VI contains the logical equations for the computer, while the final portion of the thesis, Part VII, comments on possible areas of application and useful revisions of the proposed design.

II GENERAL COMMENTS ON MICRO-PROGRAMMING

2.1 The History of Micro-Programming:

Although earlier papers (2)* may have discussed some of the concepts associated with micro-programming, the term itself was apparently first expressed by Wilkes and Stringer (3)(4). Figure 1 illustrates the micro-control unit in the scheme originally proposed by Wilkes and Stringer. A pulse is applied to the decoding tree, and is routed to one of the horizontal output lines of the tree by the configuration of Register I. The output lines of the tree are connected to the diode matrix A; the pulse on the selected horizontal line of the matrix is applied to selected vertical lines, operating gates which are associated with micro-operations. Hence, a particular number in Register I selects a corresponding line of matrix A; the connections made onto the selected line determine a particular combination of micro-operations. Therefore, each output line corresponds to a micro-order.

The selected horizontal line also causes an output from matrix B, which is used to set Register II. The setting of Register II determines the micro-order next to be performed, since the gate between the two

---

* Moore and Shapin describe the design of a proposed serial computer, where control over the arithmetic processes would be obtained from pulses on a magnetic drum. "...Each order that the machine is to perform is coded as a sequence of successive simpler gating, logical, and transfer operations on only one binary digit..." The control pulses were to be recorded more or less permanently on the drum, and would fix the order list of the computer. The operating speed of the computer was fairly slow; only one commanded sequence could be performed per drum revolution. Half the circumference of the drum was devoted to the controls, the other to the usual storage.

registers causes the contents of Register II to be transferred to Register I just before the next control pulse is applied. Therefore, pulses applied alternately to the decoding tree and the register gate cause execution of a pre-determined sequence of micro-orders.

The x connection shown on a horizontal line routes the pulse onto either of two lines of matrix B; the choice is under control of a flip-flop, and hence allows conditional branching.

Not shown are the registers associated with the arithmetic element, or portions of the control unit which are affected by the micro-operations.

Billing and Hopmann (5) describe the above scheme, and mention also that the diode matrices were replaced in a later design by magnetic core matrices, enabling more convenient modification of the micro-orders after completion of the machine. (This latter comment is verified by Bauer (6), who states that the technique was applied to EDSAC II.) Billing and Hopmann proposed a further modification, however, in which the selected line corresponds to a magnetic core shift register. The propagation of a single pulse from one stage of the shift register to the next actuates gates connected to the terminals between stages. Thus, each advance of the pulse through the shift register generates a micro-operation. For conditional commands, the shift register branches into alternate paths; the pulse is routed to a selected path under control of conditional flip-flops. The reference states that this scheme of micro-control is applied to the design of the Gottingen G-3 computer.

Papers by both Glantz (7) and Mercer (8) discuss possible terminology, describe various advantages of micro-programming, and propose possible schemes for the internal organization of computers utilizing micro-

operations.

Generally, however, except for the Moore and Shapin report (2), and a comment in the Wilkes and Stringer paper (3), the references describe the application of micro-programming techniques to parallel machines. Also, except for a sample micro-routine in the Wilkes and Stringer paper, no explicit mention of the individual micro-orders is made.

The part following describes an application of the micro-programming concept to serial digital computers, and how certain philosophical differences seem to apply in the serial and parallel cases. Later portions of this thesis describe the particular micro-commands which appear to be the most useful.

2.2 Micro-Programming Applied to Parallel and Serial Machines:

For computers operating in parallel fashion, the fundamental time unit for a synchronous machine is a pulse-time, or bit-time. The micro-control facility, therefore, must necessarily be capable of commanding micro-operations from one pulse-time to the next. Furthermore, since the individual bits of a word are processed simultaneously, each bit-position must normally be provided with gates and logic; therefore, it is feasible to adapt the micro-control to treat each bit of a word individually, should this be required. It follows, therefore, that although the implicit information content of the micro-program storage may have to be large, it is possible to perform unique and unusual operations upon the binary information.

But applying micro-control to serial computers requires a fundamental change in the micro-control process itself. Since the basic time

interval in a serial computer is closely related to a word-time (information in a word is fed serially through the processing units, with all bits except perhaps those on the ends treated uniformly), it is natural to assume that the micro-control should be capable of exerting control from one word-interval to the next. However, it would not appear that micro-control over pulse intervals, or even partial word intervals, would be warranted economically. Computers are operated serially, primarily to effect economies in computer components (at known sacrifices in computing speed), so that the design of a micro-control facility must be consistent with this philosophy.

Limiting the micro-control to word intervals has several consequences. An interesting effect is the change in scope of what would be considered a micro-operation. In a parallel machine, the micro-control during an addition process would probably determine the operation of half-adders and carry stages for each bit position; in a serial machine, the full-word addition might be defined to be a micro-operation*.

Of course, whenever an operation is intrinsically a bit-by-bit operation, the word-interval control is inefficient. Consider, for example, magnitude comparison between two binary numbers. In a serial machine with built-in magnitude comparison logic, a usual method would be to compare the corresponding binary digits of the two numbers as they passed through the arithmetic unit. The state of a flip-flop at the end of the word cycle would indicate which of the numbers was the greater. To duplicate this bit-by-bit comparison by using micro-operations commanded

---

* Further discussion of the scope of a micro-operation will be found in Part 3.1.

at word-intervals, would probably require as many word-times as digits in a number. (With word-interval control, a more natural method would be to take the difference of the numbers and note the sign. However, the above example points out the restrictions imposed by the coarser control sampling rate.)

For a majority of instances, however, the word-interval-rate for micro-operations appears to be adequate, since many of the desired operations either operate upon the entire word or require a full word interval to make available the required part of the word.

The word-interval control requires that each micro-command be executable in a word-time. Table 2 illustrates the micro-command list for the computer described in this thesis. The particular micro-commands and their relative grouping (to allow certain simultaneous micro-operations) were determined by deriving micro-routines for a variety of command sequences, and then selecting micro-commands which allowed the most convenience and versatility.

2.3 Future Application:

It would appear that the basic differences between the serial and the parallel computer affect the application of micro-programming, and indeed, its future development. For large-scale, parallel digital computers, primarily those in which speed is of paramount importance, it appears mandatory that a micro-program facility be capable of executing all operations as quickly as would be possible by a built-in command. It must further offer the advantage of extreme flexibility, allowing specific problems to be solved more efficiently than would be possible with

built-in commands.

For serial computers, and in particular, the more modest of this class, a micro-control facility might well serve its purpose if it facilitated the use of the machine. Speed of problem solution would be expected and probably achieved, but this would not necessarily be the prime motivation for including micro-programming. The decrease in programmer time, and the extension of the computer capabilities to other problem areas, would be direct consequences.

III GENERAL DESCRIPTION OF THE COMPUTER

3.1 Nomenclature:

Where possible, the terminology used, both in the general description of the computer, and in the discussion of the logical design, follows that contained in the IRE Standards on Electronic Computers (9). The list below is included, however, to emphasize differences among particular terms and to define terminology specifically applicable to the described machine. The listing is not alphabetical, but is arranged in more or less natural order; the list is sufficiently brief that no difficulties should arise.

Command: "One of a set of several signals (or groups of signals) which occurs as a result of an Instruction; the commands initiate the individual steps which form the process of executing the instruction..." In particular, a command corresponds to a specific operation performed by the computer.

Order or Instruction: Although the latter term is preferred by the cited reference, the former is used in this thesis because of its compactness. Orders form the "...artificial language for describing or expressing the instructions which can be carried out by a digital computer..." Orders contain commands, addresses, and code-bits which control the sequencing of the routines.

State: Refers to one of four major internal conditions of the computer: Idle, Fill, Compute, and Compute One-Cycle.

Phase: Refers to one of four possible conditions which usually occur in sequence. The Phases are numbered I, II, III, IV, and are general time

divisions which describe the internal operation of the computer.

Type: Refers to one of four general classes of Orders. The Type of Order determines the interpretation of the information contained in an order word.

Micro-: The prefix Micro- is used with terms such as program, order, command, etc. to denote a level of operation usually more elementary than when the terms are used without the prefix. The literature (7)(8) attempts formal definition of the terms, but it appears that the statement above suffices for the purposes of this thesis. As was pointed out in Part 2.2, the characteristics of a serial computer can be related to micro-operations which are very different from those which might be associated with parallel machines. Hence, the following terms are used to refer to a specific class of internal conditions; the prefix micro-, although it implies a more elementary class of operations, is used primarily in an organizational sense.

Micro-Command: Refers to a specific operation executed by the computer. Micro-Commands are listed in Table 2. As will be seen later, Micro-Commands are executed during Type II or Type IV orders.

Micro-Order: Consists of a specified arrangement of Micro-Commands, control digits, and addresses. Therefore, a Micro-Order can be considered as the vehicle in which the Micro-Commands are placed; the term Micro-Order should not be used to describe particular operations performed by the computer. Micro-Orders are obtained from an auxiliary core memory during the execution of Type II orders and are stored into the core facility during the execution of Type III orders.

Micro-Routine: A particular arrangement of Micro-Orders which, when executed, achieves a desired effect.

Summary of Terms: The following statements will perhaps aid in emphasizing the differences among the various terms discussed above.

During computation, the computer is always acting upon an Order. If the order is a Type I Order, then the computer will execute a Command. If the order is a Type II Order, then the computer is performing a Micro-Routine, and is obtaining, in a pre-determined manner, Micro-Orders from the core memory; it then proceeds to execute the Micro-Commands contained in the selected Micro-Orders. A Type III Order transfers Micro-Orders from the magnetic drum memory to the core memory. A Type IV Order merely causes the computer to execute the Micro-Commands present in the Type IV Order word.

3.2 Specifications:

This part of the thesis lists the general specifications of the machine. Detailed descriptions of the component units are included in Part IV of the thesis.

Internal Information: Binary.

Word Length: 32 bits per word.

Input-Output: Flexowriter, punched-tape reader, tape punch.

Magnetic Drum Memory Capacity: 63 channels, each containing 64 words, and one short storage loop with a capacity of 8 words.

Magnetic Drum Speed: Approximately 3600 rpm.

Internal Clock Rate: About 123 kilopulses per second.

Core Storage: Micro-orders are stored in a core memory containing 128 words of 24 bits each. The 24 bits of a word are read out of the core memory at a read-out frequency corresponding to word-time intervals (about 3840 per second). The core memory allows access to words at

random if required.

Arithmetic Registers: The three arithmetic registers are all one-word loops on the drum memory.

a. A Register: The main arithmetic register, and the most versatile in its information handling capabilities.

b. D Register

c. R Register

Order Counter-Register: This is also a one-word length loop on the drum, but will henceforth be referred to as the Order Counter. The Order Counter stores the address of the order normally next to be performed.

Address Register (M Register): Also a one-word loop on the drum. This register stores the address required in executing an order or a micro-routine.

B Register (Index Register): Also a one-word loop on the magnetic drum; used primarily for modifying addresses or for tally purposes.

3.3 Order Types:

A major difference between this computer and existing machines lies in the forms which can be assumed by the orders. Each order on the main memory has an associated two-digit code which places the order into one of four possible types. These four types are illustrated in Figure 2 and are briefly described below. The composition of the micro-orders is described later.

Type I Order:

This corresponds to the usual order. As shown by Figure 2, it consists of a four-digit code which specifies one of 14 commands, and 12 binary digits, the first six for channel, the latter six for sector,

which specify an address on the drum memory. The command list, shown in Table 1, has limited arithmetic capabilities; four of them relate to the B Register, and two commands relate to the operation of the short loop. All operations requiring a drum memory search for information are commanded by Type I orders.

Type II Order:

This order consists of two parts. Seven binary digits specify the core address locating the first micro-order of a micro-routine to be executed. The associated 12 bits correspond to an address on the drum, and are subsequently stored onto the Address Register. The address bits are not normally used with micro-orders. However, if a command, such as is usually associated with a Type I order, is executed from a micro-order (by use of a special code), the contents of the Address Register then perform the normal functions of the address digits in a Type I order.

Note that the seven bits which specify the start of the micro-routine can be considered as a "command code", where the "command" summarizes the results achieved by the micro-routine. Conceptually, a Type II order can be considered like a Type I order, and extends the command list as much as desired.

Type III Order:

This order transfers information from the drum into the core memory. Again, seven bits specify a location in the core storage, but in this instance, determine the starting point for storage into the core facility. The accompanying 12 digit address denotes the starting location in the drum from which the information is obtained. Subsequent words in the drum are transferred to successive storage locations in the core memory. A digit in the sign position of a word on the drum halts the

transfer process (the word containing the "End-of-Transfer" bit is the final word transferred).

Type IV Order:

This order allows micro-commands to be obtained from the drum and executed directly.

3.4 B Register (Index Register) Codes:

Several commands in the Type I command list relate to the B Register, and are described later. However, there are two other codes which use the B Register: 1) For Type I or III orders obtained from the drum, a one-bit in the sign position causes the contents of the B Register to be added to the accompanying 12-digit address. 2) Correspondingly, if a command (normally associated with Type I orders) is executed during a micro-routine (Type II order), a one-bit in column 23 of the core micro-order causes the contents of the B Register to be added to the contents of the Address Register. (The composition of a micro-order is described in 3.6.)

3.5 Description of the Commands:

Table 1 lists the commands normally associated with Type I orders. The following is a brief explanation of the commands. In all cases, the command number corresponds to the binary equivalent of the command code; the letter m refers to the accompanying 12-bit address.

002: A into C(m): (Abbreviated AM m) Stores the contents of the A Register into location m of the drum memory.

003: C(m) into D: (MD m): Transfers the information from location m of the drum into the D Register.

004: Transfer to the Short Loop: (MS m): Transfers 8 words, starting at location m on the drum memory, into the short loop (Channel 0).

005: Transfer from the Short Loop: (SM m): Transfers the contents of the short loop (Channel 0) into 8 successive word positions of the main memory, starting at location m.

006: Print: (PT m): Causes the Flexowriter to print the symbol represented by the bits in the 6 msd positions of the address code m.

007: Partial Substitute: (PS m): Inserts the 12 bits of the A Register which correspond to the address digits, into the address portion of the word in location m.

008: Extract: (EXT m): Erases the A Register, except where one-bits are present in the word in location m. Alternative view: bit-by-bit logical multiplication of the contents of A and m.

009: Input: (IN): Starts the punched-tape reader, and simultaneously places the computer into the Idle State.

010: Store the Augmented Order Counter Contents: (OCM m): Increases the Order Counter contents by one, and stores in location m.

011: C(m) into B: (MB m): Places the 12 address bits of the word in location m into the corresponding positions of the B Register.

012: Halt: (H): Places the computer into the Idle State.

013: Decrease B: (DB m): Decreases the B Register contents by one. If the resulting B value is negative, proceed normally to the next order. If B is zero or positive, proceed to the order found in location m.

014: B into A: (BA): Transfers the contents of the B Register into the A Register.

015: Increase B: (IB): Increases the B Register contents by one.

3.6 Description of the Micro-Commands:

Figure 3 shows the relative word position of a micro-order when stored on the drum, prior to transfer to the core memory. The component micro-commands in a micro-order are also shown in Figure 3, and are described below, starting with the msd or left end of the micro-order.

N Counter Code: (N - 1 Bit):

The presence of a one-bit in the msd position of the micro-order automatically decreases the contents of the N Counter (Operation Counter) by one. If this results in a value N which is positive or zero, then the next micro-order is obtained from the row position of the core memory specified by the Y_n address included in the micro-order; if the resulting N is negative, the micro-order next in line is used.

Arithmetic Micro-Commands:

The terms, Arithmetic and Control, as applied to the two major groups of micro-commands, bear a general relation to the results accomplished by the micro-commands in the two groups. However, the terms are primarily for convenience, and no strong attempts have been made to restrict the possible operations in the two groups to arithmetic or control functions. The four bits of the Arithmetic Micro-Command Code are used to represent fifteen separate micro-commands; these are numbered in the 100 series, where the last two digits correspond to the binary equivalent of the code. The Arithmetic Micro-Commands are listed in Table 2.

101: Complement A: (comp. A): Causes the contents of the A Register to be complemented (true binary complement).

102: A + D into A: (a ADA): The sum of the A and D Registers is placed into A. Sign bits are treated like the other digits.

103: A - D into A: (s ADA): Subtract the contents of the D Register from those of the A Register. Signs do not affect the operation.

104: A + D into R: (a ADR): The sum of A and D is placed into the R Register. Note that both A and D are left unchanged.

105: A - D into R: (s ADR): The contents of D are subtracted from A and placed into the R Register.

106: Clear A: (cl A): Clear the A Register.

107: Clear R: (cl R): Clear the R Register.

108: Clear D: (cl D): Clear the D Register.

109: Insert 0 into msd D: (0 msd D): Zero-sets the most significant digit (left-most) of the D Register.

110: Insert 0 into lsd R: (0 lsd R): Zero-sets the least significant digit (right-most) of the R Register.

111: Insert 1 into lsd R: (1 lsd R): One-sets the lsd of R.

112: Insert 0 into msd R: (0 msd R): Zero-sets the msd of R.

113: Insert 1 into msd R: (1 msd R): One-sets the msd of R.

114: Insert 0 into msd A: (0 msd A): Zero-sets the msd of A.

115: Insert 1 into msd A: (1 msd A): One-sets the msd of A.

Control Micro-Commands:

The four bits of the Control Micro-Command Code represent fifteen separate micro-commands; these are listed in the 200 series and are shown in Table 2.

201: Return to Main Control: (rmc): Causes the computer to execute the micro-order in which this micro-command occurs and then to return to Phase I to execute the next order, as specified by the Order Counter.

Effectively, this is an "end of micro-routine" signal.

202: Shift A Right: (sr A): Causes a right-shift of the contents of the A Register. The lsd is lost; a zero-bit is normally inserted into msd A.

203: Shift A Left: (sl A): Causes a left-shift of the contents of A. The msd is lost; a zero-bit is inserted into lsd A.

204: Circular Shift A Right: (src A): The contents of the A Register right-shift by one position; the original lsd of A is inserted into the msd position.

205: Circular Shift A Left: (slc A): The contents of the A Register left-shift by one position; the original msd of A is placed into the lsd position.

206: Shift A and R Right: (sr AR): Considers the A and R Registers as a double-length register, with A as the most significant half (the left half), and with R as the least significant half. Causes a single right-shift of the combined contents; the lsd of R is lost, and a zero-bit is placed into the msd of A.

207: Shift A and R Left: (sl AR): Similar to 206, but for the left-shift case.

208: Transfer A into the Address Register: (tr AMr): Causes the address portion of the contents of A to be transferred to the Address Register. (Mr is used to represent the Address or M Register to avoid confusion with M for memory, as used with the command code.)

209: Transfer A into R: (tr AR): Causes the contents of A to be placed into R. A is re-circulated.

210: Transfer R into A: (tr RA): Transfers the contents of the R Register into A. Re-circulates R.

211: Exchange A and R: (ex AR): Exchanges, or swaps, the contents of the A and the R Registers, so that each finally contains the original contents of the other.

212: Exchange A and D: (ex AD): Exchanges the contents of A and D.

213: Exchange R and D: (ex RD): Exchanges the contents of R and D.

214: Transfer D into N: (tr DN): Causes five bits of the D Register in positions P_6 through P_2 to be placed into flip-flops N_5 through N_1 of the N Counter (Operation Counter). For words containing orders, positions P_6 through P_0 are normally spare positions.

215: Transfer Address Register to Order Counter: (tr MrOC): Transfers the 12 bits in the address portion of the Address Register into the Order Counter.

Control Input Codes:

The three bits of this code select any of seven inputs which then determines the state of one of the Control Flip-Flops, F_3, F_2, F_1, or F_0. The codes are numbered in the 300 series; the selected Control Flip-Flop is set to the state of the selected variable.

301: lsd R: Selects the state of lsd R existing at the beginning of the word interval affected by the micro-order in which this code appears.

302: lsd A: Selects the state of lsd A at the beginning of the word interval.

303: msd A: Selects the state of msd A at the beginning of the word interval.

304: msd D: Selects the state of msd D at the beginning of the word interval.

305: Carry: (Set K) Inspects the Carry flip-flop, K, at P_{31}; hence, selects the carry occurring at the end of the word interval.

306: Set to One-State: (Set 1)

307: Set to Zero-State: (Set 0)

Control Selector Codes:

Two binary digits select which of the four Control Flip-Flops is to be set by the input chosen by the Control Input Code. The codes are numbered in the 400 series.

400: F_0

401: F_1

402: F_2

403: F_3

Y_n Control Code:

With one exception (code 507), the three bits which comprise the Y_n Control Code effect the use of the Y_n Address digits.

501: If F_0

502: If F_1

503: If F_2

504: If F_3

The indicated codes above select one of the four Control Flip-Flops. If the selected flip-flop is in the one-state, the next micro-order is taken from the row specified by the Y_n Address. Otherwise, the micro-order next in sequence is used.

Note: A Control Flip-Flop which is set by any input except 304, can be selected by the appropriate 500 series code in the same micro-order. A 305 input (Carry) cannot be selected until at least the following micro-order, however.

505: Unconditional Transfer: (ut): Causes the next micro-order to be obtained from the position specified by the Y_n address.

506: Transfer Y_n into the N Counter: (trY_nN): The five least significant digits of the Y_n Address are transferred to the N Counter.

507: Non-Micro-Order: (nmo): Denotes that the next read-out from the core is not a micro-order, but rather, is to be interpreted as a command (commands are usually associated with Type I Orders). A command stored in the core memory is shown in Figure 4. The B Control Bit is displaced. No operand address accompanies the command; the existing contents of the Address Register specify the operand location. Also, a one-bit must be coded into the position corresponding to P_{13}.

3.7 The Interaction of Simultaneous Arithmetic and Control Micro-Commands:

In many instances, it is desirable to execute both an Arithmetic Micro-Command (100 series) and a Control Micro-Command (200 series) during the same word interval. Where the micro-commands are obviously independent, or where they would not reasonably be required simultaneously, no problems arise. However, a number of reasonable combinations are considered below. Other combinations can be analyzed by referring to the detailed logical equations.

a. In all cases, Clear micro-commands take precedence.

b. Any of the digit-insert Arithmetic Micro-Commands (109 through 115) will override the effect of a Control Micro-Command in the selected digit position. For example, the simultaneous combination of (1 msd A) and (sr A) can be used to right-shift the A Register when it contains a complemented number. The (1 msd A) Arithmetic Micro-Command overrides the normal action of inserting a zero-bit in the msd position during the execution of (sr A).

c. Other cases have been considered, and are shown on Table 3.

IV DESCRIPTION OF THE LOGICAL DESIGN

1. Introduction:

The internal condition of the computer is represented by various combinations of binary variables, where each variable is capable of assuming one of two possible states. Logical equations are used to describe the future state of the machine as a function of the existing values of the binary variables.

The binary variables are mechanized in terms of flip-flop circuits, magnetized areas of the magnetic drum, magnetization of ferromagnetic core elements, switch settings, etc. The logical equations, in general, describe the inputs to the flip-flop circuits in terms of the binary variables, and serve as a complete and detailed description of the sequences within the computer.

The parts to follow describe the logical design of the machine, first by considering various functional units of the machine (drum memory, core memory, input-output, etc.) and then by discussing the logical equations themselves. The description of the logical equations is intended to serve as a guide; complete understanding of the machine operation requires a detailed study of the equations.

The complete logical equations for the machine are included in Part 6.2. In describing the various sequences, pertinent flip-flops are extracted and included with each discussion. The logical equations for these flip-flops are further shortened to include only the terms applicable to the discussion. To study the operation of each flip-flop in detail, the reader is advised to use the complete logic included in Part 6.2.

For recirculating register write flip-flops (A_{31}, M_{31}, etc.), only the logic which causes modification of the register contents is discussed. However, the terms which cause recirculation and hence retention of the register contents are included with the complete logic.

The command and micro-command codes appear in the logic as numbers (002, 102, etc.) and are left unmechanized to aid in the understanding of the logic. Efforts have been made to select the code numbers so that the corresponding binary combinations simplify the final mechanized logic; the code combinations appear much less formidable in complexity, though more obscure in meaning, in their mechanized form. (See 6.1)

4.2 Nomenclature:

Flip-Flops:

The IRE Standards on Electronic Computers (9) defines a flip-flop as follows:

> "A device having two stable states and two input terminals (or types of input signals) each of which corresponds with one of the two states. The circuit remains in either state until caused to change to the other state by application of the corresponding signal..."

Flip-Flops are the main logical elements in the computer.

The flip-flop assumed for this thesis is of the set and re-set type, as defined above.* Flip-flop outputs are denoted by capital letters,

---

* Another, less common, flip-flop circuit changes state whenever a signal is applied to the input (sometimes called a complementing flip-flop). The logical equations exhibit a somewhat different form.

where primes and non-primes represent the zero and one-states. If a flip-flop is denoted as A_1, for example, then the two states of this flip-flop are represented as A_1 and A_1'. When the variable represented by this flip-flop is defined as being true, then:

A_1 = true, one-state, on.

A_1' = false, zero-state, off.

The converse applies, also, so that if A_1' is true, A_1 is false. Representing the true and false conditions by one and zero,

$$A_1 + A_1' = 1$$

$$A_1 A_1' = 0$$

The input to a flip-flop is denoted by a lower case letter corresponding to the upper case output term. A preceding subscript of one or zero denotes the state to which the input sets the flip-flop.

A_1: $_1a_1$ = One-set terms = f(Flip-Flops and Drivers)

$_0a_1$ = Zero-set terms = g(Flip-Flops and Drivers)

where f and g are functions of the variables in the machine, and must be mutually exclusive. Note that a clock pulse is implicit in the input logic, since the computer is operated synchronously with a clock source, and flip-flop changes occur only at clock times.

A very important characteristic is the single clock-interval delay which exists between input and output of a flip-flop. The input is determined by conditions existing at the clock sampling time; if a change is caused in the state of a flip-flop, this change is not noted until

the following clock time.

Drivers:

A driver circuit is very elementary in its logical characteristics, and is a device which, without delay or storage effects, reproduces a logical equation output. In vacuum-tube circuits, cathode followers are commonly used as drivers. Drivers are mentioned here primarily to bring out the slight difference in nomenclature. Logical equations for drivers are denoted by:

$$A = f(\text{Various f.f. and drivers})$$

$$A' = g(\text{Various f.f. and drivers})$$

where f and g are logical complements of each other. Note, also, that a driver output cannot be a function of itself or its complement, because of the absence of a delay characteristic.

4.3 Referencing the Magnetic Drum Memory:

The main magnetic drum memory contains 64 channels of information storage. Sixty-three of these are "long" channel storage, where each channel contains 64 words of 32 bits each. A short loop channel, containing 8 words of storage, is referred to as Channel 0.

The drum surface is also used for a number of one-word length recirculating registers; these are listed in Part 3.2 and described in Part 4.4.

Three additional channels are used to reference the drum memory. The first is a clock channel containing 2048 permanently recorded pulses around the circumference of the drum. The clock channel supplies the

master timing signal for the computer.

The Sector Code Channel is used to supply sector information; code configurations which correspond to the binary number equivalent of the sector next to be read are coded into positions P_{12} through P_7 of a word.* The required word sector is obtained by comparing the output of the Sector Code Channel (S_c) with the sector address of the word desired. Channel selection is accomplished by means of a selector matrix, which is actuated by the channel portion of the address.

An Origin Channel, which supplies a unique reference pulse once per revolution, is used as a zero-reference for the Pulse Counter.

4.4 Recirculating Storage Registers on the Drum Memory:

Aside from the storage channels on the memory, there are six recirculating drum storage registers which are used for storing word-length information. Since these registers are basic to the operation of the machine, a short description of their operation and nomenclature is included below.

Consider the A Register, for example, to be made up of 32 flip-flops connected as a dynamic storage register, so that information is continually recirculated through the loop at clock rate. (See Figure 5.) The 32 clock pulses required to re-cycle the loop are denoted by the symbols P_0, P_1, ... through P_{31}.** Define P_0 as the time at which the

---

* The sectors need not be numbered consecutively. Frankel and Cass (10) use a scheme where sectors are interleaved, with about eight words between consecutive sector numbers.

** P_n refers primarily to particular pulse times, but is used also to label positions in a word, especially as obtained from memory. The context makes clear, however, which meaning is intended.

information in the loop is at the reference position, i.e., considering the 32 bits as a word, the bits are positioned properly when sampled by the clock at time P_0.*

Since the flip-flops are cycling continually, there is no logical change involved if a magnetic drum surface is inserted into the loop. Logically, this is indicated by the diagram in Figure 6. Note that A_{31} now corresponds to the flip-flop which drives the memory write amplifier, and A_0 is the flip-flop which represents the read amplifier output. Since the rest of the computer merely samples the information flowing between A_0 and A_{31}, there is no logical difference between the systems of Figures 5 and 6.

Figure 7 is a simple schematic representation of the drum. The actual spacing between read and write heads is, of course, modified by physical considerations such as the requirement that clocking shall occur at the center of the output signal, flux leakage, etc.

To enable relative shifting of the stored information so that at time P_0 the bits shall appear to have shifted either to the right or to the left, it is necessary either to shorten or lengthen the loop for one word cycle. Consider the following changes: 1) insert an extra flip-flop, A_{32}, which enables a one-bit delay, 2) decrease the head spacing by one-bit, substituting another flip-flop A_1 to compensate. Note now, the three possible paths for information flow (Figure 8). By using a loop which is one-bit shorter than the 32 bit word length, information is effectively shifted to the right (or ahead in time) by one position. On

---

* Although P_0 is the actual reference time, P_{31} is of more interest, since the logic represents inputs to flip-flops, and the control over a word interval is specified at P_{31} of the preceding interval.

the other hand, a loop which is increased by one causes a shift left (or delay in time) to occur. The void which is left when a shift occurs can be filled as desired, and examples of this will be noted later when specific shift operations are described.

4.5 Operation of the Core Memory:*

Organization:

A modest magnetic core memory with a capacity of 128 words, each word 24 bits long, is provided to store and control the micro-routines. Using a Type III order, information from the magnetic drum memory can be transferred into the core memory as desired.

Consider the core arrangement as a planar matrix where each of the rows stores a micro-order; the cores are arranged with 24 columns and 128 rows. The desired row is selected by the 128 possible configurations of the Row Select Register, Y_7 through Y_1.

Read-out from the core memory selects an entire row, or word, so that no column selection is required. Usual random access magnetic core memories are often considered as stacked planes, in which each plane represents a digit position; x and y selection within the planes then enables the read-out of a bit from each of the planes, resulting in the simultaneous availability of all of the bits of a word. The core memory described here corresponds to just one of the planes of the usual random-access memory, but with the further simplification that no x selection is

---

* References (11)(12)(13), among others, describe techniques using magnetic cores which can easily be adapted to achieve the requirements described here.

required. Hence, the core facility is a relatively simple one.

Read-Out From the Core Memory:

Information is obtained from the core storage only once per word interval. Therefore, the frequency response requirements are very relaxed; the read sampling rate is about 3840 per second. Because of the low operational speed, it would be feasible to use multiple turns to achieve the required ampere-turn drive; hence, low current drivers can be used.

The sequence for reading information from the core memory is as follows:

a. Read-out from the core* occurs at P_{31}, so that information stored in the selected row is available to control the operation of the subsequent word interval.

b. Regeneration of the row from which the information was obtained is denoted in the logical design as the P_{15} time following each read-out.** It is expected that the regeneration process can utilize auxiliary cores which are set at the time of read-out. Should this be difficult to accomplish circuit-wise, flip-flops can be used.

c. At P_{28}, the Row Select Register, $Y_7 - Y_1$, is set to the address of the row next desired.

The read-out signal is $M_m T_2' P_{31}$, which is true only for Type II orders. The regeneration signal is similar, except with P_{15} in place

---

* Henceforth in this thesis, the term core, alone, will be used occasionally in referring to the core memory.

** The exact regeneration time is not important. P_{15} was chosen because it falls midway between the P_{31} read-out times, and hence would minimize the frequency response requirements of the row selector circuits.

of P_{31}.

Filling the Core Memory:

To fill the core, as occurs during Type III orders, information from the drum is fed into a shift storage register during the interval P_{27-0}. (Actually, only P_{27-4} is required, but P_{27-0} is more easily mechanized.) Again, cores can probably be used for this purpose. At P_0, the contents of this register are read into the row selected by Y_7 through Y_1. The fill signal during Type III operation is denoted by $M_m T_1 P_0$.

4.6 Input-Output:

This section describes one of numerous feasible methods for interconnecting the computer to a Flexowriter and punched-tape reader. Expanding the input-output capabilities to include supplementary Flexowriters, punched-card readers, magnetic tape units, etc. is primarily a matter of superposing additional tie-in capabilities to the present design and providing for the possibly differing speed capabilities of the various units. The proposed scheme is very conservative, in that logic is used to provide proper timing during the input and output sequences; the logic could be simplified by using the response characteristics of circuit elements to provide the desired control.

The main buffer element between the computer and its external equipment is the Input-Output Register, H_6 through H_1. The six flip-flops of this register correspond to the assumed number of channels in the paper tape; extension to eight channel tape operation would require only that flip-flops be added to the Input-Output Register. During input, signals due to the presence of holes in the channels are used to one-set the corresponding H flip-flops. During output, the H flip-flops are set by the

computer, and drive either relays or thyratrons which then actuate the Flexowriter key mechanism.

The comments below apply to the logic associated with the timing of the input and output operations. The manner in which the computer assimilates input information is described in Part 4.14, which outlines operation during the Fill State. The way that output information is provided is discussed in Part 4.12, which describes the operation of the Print Command (006).

Input Timing:

During the input process, the presence of a hole in a channel of the punched-tape causes the corresponding H flip-flop to be one-set. At the same time, a signal from the tape reader sprocket channel (TRP: Tape Reader Pulse) is used to one-set the I_1 flip-flop. The one-state of the I_1 flip-flop thus indicates that an input signal has been received, but that, as yet, the information has not been assimilated into the computer.

The one-set terms for I_1 and the end stages of the H Register are shown below. Note that the signals from the tape reader are not in synchronism for all channels, because of mechanical differences, and are also subject to contact bounce transients.

$$I_1: \quad {}_1i_1 = (\text{TRP})I_1$$

$$H_6: \quad {}_1h_6 = (\text{Channel 6})$$

$$H_1: \quad {}_1h_1 = (\text{Channel 1})$$

The lifting of the tape reader contacts to their retracted position causes the term TR_b to become true (TR_b implies the "back" contacts of the tape reader, and is formed by the series connection of switches, one

for each channel, which close when the reader contacts are in the retracted position. TR_b could also be the single back contact on the sprocket channel.). Since TR_b is true when all of the reader contacts are retracted, this assures that input signals to the H Register have been terminated.

Subsequently, flip-flop I is turned on by the logic shown, and remains on for one word interval. The one-state of flip-flop I initiates and controls the actual transfer of the H Register contents to the rest of the computer (see Part 4.14). Also, the true state of flip-flop I zero-sets I_1, so that the input sequence can be initiated but once per input from the tape reader. The H Register is also zero-set by flip-flop I, since the assumption is made here that the tape reader contact signals can be used only to one-set the H flip-flops.

Flip-flop I turns on only for allowable tape code combinations. This allows blank tape, or punched combinations representing code delete, tab, etc., to be fed through the tape reader without affecting the computer. Non-allowable combinations directly zero the H Register, since flip-flop I cannot be used in these instances.

$$I_1: \quad {}_1i_1 = (TRP)I_1'$$

$$\quad {}_0i_1 = I_1(TR_b)IP_{31}$$

$$I: \quad {}_1i = I_1(TR_b)(TP_b)\ (\text{Allowable H Combinations})P_{31}$$

$$\quad {}_0i = IP_{31}$$

$$H_6: \quad {}_0h_6 = IP_{31} + (TR_b)\ (\text{Non-Allowable H Codes})$$

The punched-tape reader can be activated either manually or by the

Input Command (009) in a program. In the latter case, the tape reader must contain the desired tape in ready position.

Output Timing:

During the execution of the Print Command (006), as discussed in Part 4.12, the H Register is set to the desired code configuration. The End-of-Command Signal for this command, $GU_1(006)P_{31}$, also triggers the punch and type mechanism of the Flexowriter. The I_1 and I flip-flops are used in the same manner as in the input process, except that the signals now originate from the tape punch mechanism, rather than from the tape reader (hence, TPP and TP_b, rather than TRP and TR_b). The one-state of the I flip-flop now indicates that the particular output operation has been completed, and that the H Register is ready to receive the next signal.

The logic below is similar to that for the Input Timing, except that TPP and TP_b are used. Term J is an added input to the one-set of the I flip-flop, so that the I flip-flop will be turned on during the output process regardless of the H Register configuration.

I_1: $\quad {}_1i_1 = (TPP)I_1$

$\quad\quad {}_0i_1 = I_1(TP_b)IP_{31}$

I: $\quad {}_1i = I_1(TR_b)(TP_b)JP_{31}$

$\quad\quad {}_0i = IP_{31}$

4.7 Operation During Phases I and II:

The phase condition of the computer is denoted by flip-flops U_2 and

U_1. These serve as a major timing control on the internal operation of the machine. Assuming that the computer is in the active computation state, Table 4 illustrates, very generally, the sequences followed in performing the various order types. During Phases I and II, the operation of the machine is independent of the type of order; branching occurs at the beginning of Phase III, under control of T_2 and T_1, which denote the type of order.

This part of the thesis describes the operation during Phases I and II, common to all of the order types.

Phase I Operation: Order Search:

During Phase I($U_2'U_1'$), the computer searches the drum memory for the order to be executed. To accomplish this:

a. W_c is one-set at the beginning of each word interval.

b. The output of the Order Counter (O_0) and the signal from the Sector Address Channel (S_c) are compared. If these disagree, W_c is zero-set.

c. If W_c is true at P_{31}, this indicates that the following word-interval is the desired interval. The transition is made to Phase II.

$$W_c: \quad {}_1W_c = V_1U_2'U_1'P_{31}$$

$${}_0W_c = U_2'U_1'(O_0S_c' + O_0'S_c)(Ch_0P_{9-7} + Ch_0'P_{12-7})$$

$$U_1: \quad {}_1U_1 = U_1 W_c P_{31}$$

U_2' is not required in the one-set to U_1, since this same term is later used to control a transition from Phase III to Phase IV after a successful search for an operand.

The term V_1 in the one-set of W_c denotes that the computer is in the

active computational state; this permits the machine to execute the search and consequently to progress out of Phase I.

The Ch_0 term in the zero-set of W_c indicates that the desired instruction is in Channel 0, the eight-word loop. For Ch_0, the sector comparison needs to extend only over the interval P_{9-7}, since the loop contents are numbered modulo-eight; for all other channels, the entire sector address interval, P_{12-7}, is required.

Ch_0 and Ch_0' are derived from the setting of the Channel Select Register, C_6 through C_1, which stores the code representing the desired channel. During Phase I, C_6 through C_1 sample the channel portion of the address stored in the Order Counter. The first possible search is delayed for one word interval by setting W_c to the zero-state just before the computer enters Phase I (the logic is not shown here). Hence, W_c cannot be one-set until one word interval has elapsed during Phase I, enabling C_6 through C_1 to obtain the proper channel code. (The channel value is important during the sector search only to differentiate between the short-loop channel and the other channels.)

During Phase I, C_6 through C_1 are connected as a shift register. Only the end stages are shown here. After transition to Phase II, the channel address contents are frozen to control the actual transfer of the word from main memory.

$$C_6: \quad {}_1c_6 = U_2' U_1' {}^0_0 P_{18-7}$$

$$\quad {}_0c_6 = U_2' U_1' {}^0_0' P_{18-7}$$

$$C_1: \quad {}_1c_1 = U_1' C_2 P_{18-7}$$

$$\quad {}_0c_1 = U_1' C_2' P_{18-7}$$

Phase II Operation: Preperation for Order Execute:

During Phase II ($U_2'U_1$), which is true only for one word interval, the following operations occur:

a. The contents of the Order Counter are increased by one, under command of O_c.

b. The selected instruction word is transferred from the memory to the Address Register or M Register.

c. The bits in positions P_{26} through P_{19} of the selected order word are transferred into flip-flops $Q_4Q_3Q_2Q_1S_4S_3S_2S_1$ respectively. In the event that the instruction is a Type I order, the contents of S represent the desired Type I command. If the instruction is Type IV, then Q and S will contain the Arithmetic Micro-Command and Control Micro-Command respectively.

d. The bits in positions P_{25} through P_{19} of the selected instruction word are stored in the Y Register, which selects the read-out row from the core storage facility. Hence, should the selected instruction be either Type II or III, the proper core address is available.

e. The contents of positions P_{28} and P_{27} are stored in T_2 and T_1. These denote the Order Type.

f. W_c is set to zero except for Type IV orders (see comment in the section describing Type IV).

$$O_{31}: \quad 1^{O}_{31} = U_2'U_1(O_cO_0' + O_c'O_0)$$

$$0^{O}_{31} = U_2'U_1(O_cO_0 + O_c'O_0')$$

$$O_c: \quad 1^{O}_{c} = U_2'U_1'W_cP_{31}$$

$$0^{O}_{c} = U_2'U_1O_cO_0'P_{18\text{-}7}$$

M_{31}: $1^{m}_{31} = U_2'U_1W$

$0^{m}_{31} = U_2'U_1W'$

Flip-flops $T_1Q_4Q_3Q_2Q_1S_4S_3S_2S_1$ are connected as a continuous shift register, where the output from the drum, W, enters T_1 and is shifted to the right. T_1 is included in the shift chain, since this enables the use of interval P_{27-0}, allowing simpler mechanization of the shift interval. Only the flip-flops T_1, Q_4, and S_1 are shown below:

T_1: $1^{t}_{1} = U_2'U_1WP_{27-0}$

$0^{t}_{1} = U_2'U_1W'P_{27-0}$

Q_4: $1^{q}_{4} = U_2'U_1T_1P_{27-0}$

$0^{q}_{4} = U_2'U_1T_1'P_{27-0}$

S_1: $1^{s}_{1} = U_2'U_1S_2P_{27-0}$

$0^{s}_{1} = U_2'U_1S_2'P_{27-0}$

Y_7 through Y_1, which comprise the Row Select Register for the core memory, are connected here as a shift register, with the information from W entering Y_7 and then shifted to the right. Only the end stages, Y_7 and Y_1 are noted here:

Y_7: $1^{y}_{7} = U_2'U_1WP_{25-0}$

$0^{y}_{7} = U_2'U_1W'P_{25-0}$

Y_1: $1^{y}_{1} = U_2'U_1Y_2P_{25-0}$

$0^{y}_{1} = U_2'U_1Y_2'P_{25-0}$

$$T_2: \quad {}_1t_2 = U_2'U_1WE_2'E_1'$$

$$\quad\quad {}_0t_2 = U_2'U_1W'E_2'E_1'$$

$$W_c: \quad {}_0W_c = U_2'U_1(T_2' + T_1')P_{31}$$

$$U_2: \quad {}_1u_2 = U_2'U_1P_{31}$$

The shift intervals for the Q, S, and Y flip-flops are longer than the intervals during which valid information is available, but since only the final values remaining in these flip-flops are of consequence, the wider shift intervals were used to enable slightly simpler logic. T_2 is properly set, since the final $E_2'E_1'$ condition occurs at P_{28}, the required sample time.

The transition is made to either Phase III or Phase IV, depending upon the order type (these variations are described in later sections). U_2 is always set to the one-state at the end of Phase II, since this provides for either possibility.

4.8 Operation During Phases III and IV: Type I Order:

If, at the end of Phase II, the Order Type flip-flops have the configuration $T_2'T_1'$, this denotes that a Type I Order is to be executed. The contents of $S_4S_3S_2S_1$ are to be interpreted as a command. (Driver G causes the machine to interpret the contents of S as a command code.) The following parts discuss the internal events occurring during Phases III and IV for a Type I Order.

Generally, the actual execution of a command occurs during Phase IV. Phase III is primarily concerned 1) with modifying the contents of the

Address Register, if so commanded, by adding the B Register contents, and 2) with searching the memory should an operand be required.*

Phase III: (Type I):

During and after entry into Phase III (U_2U_1'), the following operations are performed:

a. If the contents of the B Register are to be added to the contents of the Address Register, the addition occurs during the first word interval of Phase III, under command of O_c. O_c is one-set by the B-bit code in F_{31} of the Address Register.

b. The Channel Select Register, C_6 through C_1, receives the channel address portion of the word held in the Address Register.

c. Commands requiring a memory search for the operand cause W_c to be one-set. W_c is used to compare the Sector Code Channel (S_c) and the Address Register (M_0). It is required, even with a B modification, that the proper channel address be available for a word cycle before W_c is one-set, to prevent improper search due to a residual value in C_6 through C_1. By causing W_c to be in the zero-state upon entry into Phase III, an automatic one-word delay occurs before it can be turned on; a B modification cycle causes an additional word delay, since O_c' is included in the one-set of W_c.

d. If W_c remains on at P_{31}, indicating that the following sector word is the desired one, the transition from Phase III to Phase IV occurs.

---

* Commands 013, 014, and 015 do not use the Phase III interval, and could conceivably skip directly from Phase II to Phase IV. However, it is convenient to have the machine proceed through all phases in sequence, since later descriptions show that a command, when executed during a Type II order, always starts in Phase III.

e. Depending upon the particular command, various other events take place during Phase III. These are discussed later with the specific sequences associated with each command.

M_{31}: $1^{m}_{31} = U_2U_1'O_c(B_0'M_0'K + B_0'M_0K' + B_0M_0'K' + B_0M_0K)P_{18-7}$

$0^{m}_{31} = U_2U_1'O_c(B_0M_0K' + B_0M_0'K + B_0'M_0K + B_0'M_0'K')P_{18-7}$

K: $1^{k} = U_2U_1'B_0M_0P_{18-7}$

$0^{k} = U_2U_1'B_0'M_0' + P_{31}$

O_c: $1^{o}_{c} = U_2'U_1T_1'M_0P_{31}$

$0^{o}_{c} = U_2'U_1(M_0' + T_1)P_{31} + GU_1'O_c(013)'(015)'P_{31}$

C_6 through C_1 are connected as a shift register, with the Address Register output (M_0) entering C_6 and then shifted to the right. Only the end stages are shown below.

C_6: $1^{c}6 = U_2U_1'M_0P_{18-7}$

$0^{c}6 = U_2U_1'M_0'P_{18-7}$

C_1: $1^{c}1 = U_1'C_2P_{18-7}$

$0^{c}1 = U_1'C_2'P_{18-7}$

W_c: $1^{w}_{c} = GU_1'(002 + 003 + 004 + 005 + 007 + 008 + 010 + 011)O_c'P_{31}$

$0^{w}_{c} = U_2U_1'(M_0S_c' + M_0'S_c)(Ch_0P_{9-7} + Ch_0'P_{12-7})$

U_1: $1^{u}1 = U_1'W_cP_{31} + GU_1'(006J' + 013 + 014 + 015)P_{31}$

$0^{u}1 = (T_2' + T_1')U_2'U_1P_{31}$

The zero-set term for U_1 causes transition from Phase II to Phase III, since U_2 is simultaneously set to the one-state. (For Type IV orders, the zero-set for U_1 is ineffective, causing a direct transition from Phase II to Phase IV. This is discussed later.)

The adder input to M_{31} generates the sum of the B Register output (B_0) and the Address Register output (M_0) whenever 0_c is true. The addition takes place during P_{18-7}, the address interval. For clock times greater than P_{18}, the final setting of M_{31} is written into the Address Register, but does no harm.

The carry flip-flop K is time-shared and is also used for other addition operations. K is always zero at the beginning of an addition cycle.

The B bit, which occurs in the msd position of the order in the Address Register, causes 0_c to be one-set at entry into Phase III. The term T_1' allows B addition only for Type I and III operation; Type II has a B addition operation only if a command is executed.

The zero-set of 0_c has the term $U_2'U_1(M_0' + T_1)$ to insure that 0_c is zero in Phase III except for the B addition case. Note that the operation of adding one to the Order Counter during Phase II normally zeroes 0_c, except when the Order Counter contents are 11111.

The second term in the zero-set to 0_c causes 0_c to be set to zero after one word cycle in Phase III except for commands 013 and 015. (Later descriptions show that these two commands require 0_c to be in the one-state when making the transition to Phase IV.)

Flip-flops C_6 through C_1 receive the channel address from the Address Register, and are subsequently frozen during Phase IV when actual channel selection occurs.

The series of commands* in the one-set to W_c represent those which require a memory search. The comparison of M_0 and S_c extends over a shorter interval for the short loop (Channel 0), but encompasses the entire sector address (P_{12-7}) for all other channels. When a successful search ensues, as denoted by $W_c P_{31}$, the logic in the one-set of U_1 causes the transition from Phase III to Phase IV.

For non-search commands 013, 014, and 015, U_1 is automatically one-set after one word-cycle of Phase III. For the Print Command, 006, the computer cannot proceed to Phase IV unless J is in the zero-state (see details of command 006).

Phase IV: (Type I):

Details of the machine operation during the Phase IV interval are better described when the individual commands are discussed, since the internal sequences are direct functions of the various commands. The completion of each command sequence generates an "end-of-command" signal, causing the computer to return to Phase I and begin the search for the next order. The "end-of-command" signals are included with the descriptions of the individual command sequences.

Flip-flop W_c is zero-set during Phase IV, so that it is zero at entry into Phase I.

$$W_c: \quad {}_0W_c = GU_1 W_c P_{31}$$

---

* Command codes are represented by the configuration of the S flip-flops, multiplied by term G. G is the output of the Command Execute driver, and is true for Type I orders, and for the "Non-Micro-Mode" operation of a Type II order.

4.9 Operation During Phases III and IV: Type II Order:

The operation of the computer during Phases III and IV for a Type II order is closely related to the operation of the core facility. As a result of the Phase I and II sequences, the Row Select Register, Y_7 through Y_1, contains the address of the row from which the first micro-order is to be read. If there is an operand address involved, the address is already present in the Address Register, still subject to a possible modification by the B Register.*

Micro-orders, read out from the core storage at word intervals, are executed during Phase IV. Details of the logic for the individual micro-commands are described later (Part 5.13). The descriptions accompanying this section concern themselves with the following areas: 1) the transition into the "Micro-Mode" phase, i.e., the interval during which micro-commands are executed, 2) the return to Phase I when the desired micro-routine is finished, 3) the transition to a pseudo Type I operation, whenever it is required to execute a command from the core,** 4) the return transition to "Micro-Mode" after performing a Type I command from the core.

Previous comments on the operation of the core storage have indicated the following timing in the operation of this memory. Whenever $M_m T_2'$ is true, the contents of a row, selected by Y_7 through Y_1, are read out from the core at P_{31}. Hence, the bits stored in the selected row are available for use during the entire subsequent word interval. At P_{15} of the

---

* The Address Register Contents are associated with the execution of commands, not micro-commands, and therefore are applicable only to the execution of a command from the core.

** Commands 009 and 012 should not be obtained from the core memory.

word interval immediately following the read-out, the row contents are regenerated, since the assumption is made that destructive read-out processes are used. At P_{28}, the Y Register either advances by a count, or, for a conditional transfer, receives certain digits. The net effect is that at the following P_{31} time, Y_7 through Y_1 contain the new address of the row to be read out. (Except for the P_{31} requirement on the read-out time, the other pulse times are arbitrary, as long as regeneration occurs before modification of the Y contents.)

Normal Transition from Phase II to III to IV: (Type II):

At the end of Phase II, the Row Select Register (Y_7 through Y_1) contains the order digits stored in positions P_{25-19}. The Order Type flip-flops are in state $T_2'T_1$. The machine enters Phase III for one word-time, and then enters Phase IV.

a. A transition from $U_2'U_1$ to U_2U_1', and subsequently to U_2U_1, occurs.

b. M_m is one-set at $P_{30}U_2'U_1$ so that it will be true at P_{31} just before entry into Phase IV, enabling read-out. Therefore, the first word-time in Phase IV is available for computation. Note that the original Y Register setting is used for the initial read-out, since M_m is one-set late enough not to affect the contents of the Y Register for the first read-out cycle.

$$U_2: \quad {}_1u_2 = U_2'U_1P_{31}$$

$$U_1: \quad {}_1u_1 = T_2'T_1M_mU_1'P_{31}$$

$$\quad {}_0u_1 = T_2'U_2'U_1P_{31}$$

$$M_m: \quad {}_1m_m = T_2'T_1U_2U_1'X_{13}'P_{30}$$

W_c: $\quad {}_1W_c = M_m T_1 X_{13}' P_{31}$

G_m: $\quad G_m = M_m T_1 U_1$

The term X_{13}' in the one-set of M_m prevents a spurious turn-on of M_m when the computer is in Phase III, Type II, but is executing a command. The use of X_{13} will be made apparent later when the operation of a command executed from core is described.

W_c is turned on just before entry into Phase IV, and is continually one-set at P_{31} during the Micro-Mode interval. This allows proper execution of Arithmetic Micro-Command 101.

The driver term G_m implies Phase IV operation for Types II and IV and defines the interval during which micro-commands are performed. G_m causes the contents of the Q and S flip-flops to be interpreted as micro-commands.

Return to Phase I: (Type II):

Except for the case where a command is executed from the core (the logic for which is described later), the computer obtains micro-orders from the core memory, one per word-time, and executes them. To terminate a micro-routine, the Control Micro-Command 201, titled "Return to Main Control", is used.

Logical equations which apply to the return to main control (or, return to Phase I) are noted below:

a. The code for micro-command 201 is sensed at P_{31} and causes U_2U_1 to change to $U_2'U_1'$.

b. Simultaneously, 0_c is turned on.

c. At P_{16} of the following word interval, 0_c causes M_m to be

turned off.

d. W_c is set by micro-command 201 so that it is zero at entry into Phase I.

$$U_2: \quad {}_0u_2 = G_m(201)P_{31}$$

$$U_1: \quad {}_0u_1 = G_m(201)P_{31}$$

$$O_c: \quad {}_1o_c = G_m(201)P_{31}$$

$$M_m: \quad {}_0m_m = M_m T_2 {}'O_c P_{16}$$

$$W_c: \quad {}_0w_c = G_m(201)P_{31}$$

The code which corresponds to Control Micro-Command 201 is part of a read-out from the core occurring at P_{31}. However, since 201 is not sensed until the following P_{31}, all of the codes contained in the same micro-order as 201 are acted upon. At P_{31} of the final word interval in micro-mode, U_2U_1 is changed to $U_2'U_1'$. Therefore, although M_m is still true, no action is taken upon the final read-out (the word following the micro-order containing 201), since the transition from Phase IV to Phase I has caused G_m to become false.

O_c is used to turn M_m off at P_{16} of the following word. This insures that the final read-out from the core, although not used for the micro-routine, is regenerated, preventing the loss of this information. O_c does not have a zero-set, and consequently remains on into the subsequent Phase I interval. However, this does not do any harm.

Command Execution During Type II:

The execution of a command from a Type II order is initiated by the presence of a 507 code in the Y_n Control Code section of a micro-order in

the core. This code indicates that the micro-order next obtained from the core is to be interpreted, not as a combination of micro-commands, but rather, as a command, such as occurs during Type I orders. The 507 code is literally to be interpreted as "the next read-out from the core is to be considered as a command, not a micro-order". (The appearance of a command code stored in the core is shown in Figure 4.)

The following sequence ensues. The computer returns to Phase III, and M_m is set to the zero-state. Thereafter, all operations are identical to those which occur during Phases III and IV for a Type I order, except that at the completion of the command, the computer returns, not to Phase I as would be the case for a Type I order, but to the Micro-Mode. During the execution of a Type II order, the computer can revert to Phase I only by executing the 201 micro-command previously described.

At P_{31}, assume that a micro-order containing the code 507 has been obtained from the core. By means of logic not described here (the equations are described later in Part 4.13), X_{13} is made to be true at P_{31} of the following word cycle. This indicates that the configuration being read from the core at that time represents a command. Hence, the following occurs:

a. M_m is set to zero at P_{31}.

b. Simultaneously, U_2U_1 makes the transition to U_2U_1', so that the machine will be in Phase III.

c. If a B Register modification of the Address Register contents is to be made, a one-bit will be present in the P_{23} position of the word containing the command code. (This is equivalent to the code for Arithmetic Micro-Command 101.) This core output is denoted by z_{23}, and is used to one-set O_c at P_{31}. Otherwise, O_c remains in the zero-state

resulting from Phase II.

d. The inclusion of a code bit in the core position corresponding to flip-flop X_{13} insures that X_{13} remains on during the read-out of the micro-order containing the command code. This prevents M_m from turning on during the Phase III interval associated with command execution.

e. X_{13} is zero-set during Phase IV, so that, at the subsequent return to Micro-Mode operation, X_{13}' will allow W_c to be one-set.

f. W_c is zero-set to duplicate the normal entry into Phase III for a Type I order.

$$M_m: \quad 1^{m}_{m} = T_2' U_1 U_2 U_1' X_{13}' P_{30}$$

$$0^{m}_{m} = G_m T_2' X_{13} P_{31}$$

$$U_1: \quad 0^{u}_{1} = G_m T_2' X_{13} P_{31}$$

$$0_c: \quad 1^{o}_{c} = G_m T_2' X_{13} z_{23} P_{31}$$

$$X_{13}: \quad 0^{x}_{13} = G U_1$$

$$W_c: \quad 0^{w}_{c} = G_m X_{13} P_{31}$$

Except for flip-flops T_2 and T_1, the machine is in exactly the internal condition which occurs at the transition into Phase III during a Type I order. Therefore, comments describing Phase III and IV operation of a Type I order are applicable.

As before, 0_c controls the B addition. Flip-flops $S_4 S_3 S_2 S_1$ contain the code designating the desired command. If an operand is required, the address is specified by the existing contents of the Address Register; the command from core does not carry an address.

The micro-order containing the command code has not yet been regenerated, because M_m was zero-set at P_{31}, the read-out time. However, regeneration occurs when the computer finishes executing the command, and returns to Micro-Mode operation.

To return to the Micro-Mode state after executing a command from core, the following events must occur:

a. The various "End-of-Command" signals associated with the separate commands are used to set M_m to the one-state at P_{31}.

b. Simultaneously, the computer returns to Phase III, rather than to Phase I as would be the case for a Type I order. This is accomplished by causing U_1 to be zero-set by the "End-of-Command" signals, independent of order type, but by allowing U_2 to be set to zero only for a Type I order.

$$M_m: \quad {}_1{}^m{}_m = GT_1[002 + 003 + (004 + 005)M_4' + 006 + 007 + 008 + 010 + 011 + 013(0_c + W_c) + 014 + 015]U_1P_{31}$$

$$U_2: \quad {}_0{}^u{}_2 = GT_1'[\quad " \quad " \quad]U_1P_{31}$$

$$U_1: \quad {}_0{}^u{}_1 = G[\quad " \quad " \quad]U_1P_{31}$$

(The ditto marks in the brackets represent the same series of command codes as in ${}_1{}^m{}_m$.)

When returning to Micro-Mode operation, M_m is turned on at P_{31} and hence is true at P_{15}, causing regeneration of the information read out at P_{31} just prior to transition into the "Non-Micro-Mode" condition. Therefore, although the digression to perform a command from core may have imposed a lengthy delay between read-out at P_{31} and regeneration at some

P_{15} later, no information has been lost from the core.

Since the computer is initially set to Phase III, M_m alone does not yet allow the computer to operate in the micro-mode. The Row Select Register is advanced at P_{28}, read-out occurs at P_{31}, and simultaneously, the computer is placed into Phase IV. Thereafter, normal micro-mode operation proceeds.

$$U_1: \quad {}_1u_1 = T_2 \cdot T_1 M_m P_{31}$$

4.10 Operation During Phases III and IV: Type III Order:

The performance of a Type III order causes a direct transfer of information from the drum to the core memory. At the beginning of Phase III, the Address Register contains the address specifying the start of information pick-up from the drum, while the Row Select Register, Y_7 through Y_1, holds the code corresponding to the initial row of the core storage into which the information is to be placed.

During Phase III, a memory search is conducted to find the drum information; when the search is successful, the machine changes to Phase IV, and the information transfer ensues. The Row Select Register is advanced each word interval as succeeding words from the drum memory are read into the core facility. An "End-of-Transfer" code terminates the procedure, and the machine returns to Phase I.

Phase III: (Type III):

In Phase III, the following operations are required:

a. If a B modification of the Address Register is to be made, O_c is set to the one-state at the transition from Phase II to Phase III. This is identical to the procedure followed for Type I orders. O_c is then

zero-set after one word cycle.

b. W_c is again used to compare the contents of the Address Register (M_0) and the Sector code (S_c). All comments which apply to the search procedure during Phase III for a Type I order apply here, also.

c. When the search is successful, as denoted by $W_c P_{31}$ during Phase III, the transition to Phase IV is made.

$$^{0}c: \quad {}_{1}{}^{0}c = U_2 'U_1 T_1 'M_0 P_{31}$$

$$\quad {}_{0}{}^{0}c = U_2 U_1 'T_2 {}^{0}c P_{31}$$

$$^{W}c: \quad {}_{1}{}^{W}c = T_2 T_1 'U_2 U_1 '{}^{0}c 'P_{31}$$

$$\quad {}_{0}{}^{W}c = U_2 U_1 '(M_0 S_c ' + M_0 'S_c)(Ch_0 P_{9-7} + Ch_0 'P_{12-7})$$

$$U_1: \quad {}_{1}U_1 = U_1 'W_c P_{31}$$

Phase IV: (Type III):

Information is written into the core during Phase IV. The discussion below summarizes a feasible method.

a. The Phase IV condition, $U_2 U_1$, causes information from the selected memory channel to be fed into a shift register. This register can use either the same or similar units as used for regeneration storage during Type II orders. The logic is not shown here.

b. M_m is one-set at P_{31} of the first word interval of Phase IV, i.e., at the end of the first word interval.

c. Information is written into the core at $M_m P_0$.

d. As before, M_m causes the Row Select Register, $Y_7 - Y_1$, to be advanced at P_{28}. Only the end stages are shown here.

e. The "End-of-Transfer" code bit, which is in the msd position of

the final word to be stored into the core, causes O_c to be one-set at P_{31}.

f. Simultaneously, the "End-of-Transfer" signal causes the transition from Phase IV to Phase I.

g. W_c is zero-set by the "End-of-Transfer" code for proper entry into Phase I.

h. M_m, however, remains on for two clock-times, and is zero-set at P_1 by O_c.

$$M_m: \quad {}_1m_m = T_2T_1'U_2U_1O_c'P_{31}$$

$$\quad {}_0m_m = M_mT_1'O_cP_1$$

$$Y_7: \quad {}_1y_7 = M_mT_1'Y_7'Y_6Y_5Y_4Y_3Y_2Y_1P_{28}$$

$$\quad {}_0y_7 = M_mT_1'Y_7Y_6Y_5Y_4Y_3Y_2Y_1P_{28}$$

$$Y_1: \quad {}_1y_1 = M_mT_1'Y_1'P_{28}$$

$$\quad {}_0y_1 = M_mT_1'Y_1P_{28}$$

$$O_c: \quad {}_1o_c = T_2T_1'U_1M_mWP_{31}$$

$$U_2: \quad {}_0u_2 = T_2T_1'U_1M_mWP_{31}$$

$$U_1: \quad {}_0u_1 = T_2T_1'U_1M_mWP_{31}$$

$$W_c: \quad {}_0w_c = T_2T_1'U_1M_mWP_{31}$$

Since M_m is not one-set until the end of the first word-interval in Phase IV, no information is written into the core, nor is the count of the Row Select Register advanced, until the first word has been

obtained from the drum memory.

In the transition back to Phase I, 0_c is used to delay the zero-set of M_m long enough to enable the core to receive the final word of the series. 0_c remains on, into Phase I, but this is not harmful. (The sequence here is very similar to that involved in returning to Phase I at the end of "Micro-Mode", Type II.)

4.11 Operation During Phases III and IV: Type IV Order:

In performing a Type IV order, the computer enters the Micro-Mode for one word-interval, so that the codes obtained from the drum memory during Phase II and placed into the Q and S flip-flop registers are interpreted as Arithmetic and Control micro-commands. Neither a search of the drum for operands, nor reference to the core memory, are required.

a. The computer skips Phase III entirely, and proceeds directly from Phase II to Phase IV.

b. Simultaneously, M_m is one-set.

c. Both Phase IV and M_m remain true for one word interval, so that the micro-commands represented by Q and S can be executed.

d. W_c is zero-set at the end of Phase IV.

$$U_2: \quad {}_1u_2 = U_2'U_1P_{31}$$

$${}_0u_2 = G_mT_2P_{31}$$

$$U_1: \quad {}_0u_1 = (T_2' + T_1')U_2'U_1P_{31} + G_mT_2P_{31}$$

$$M_m: \quad {}_1m_m = T_2T_1U_2'U_1P_{31}$$

$${}_0m_m = G_mT_2P_{31}$$

W_c: $_0W_c = G_m T_2 P_{31}$

To enable the proper execution of Arithmetic Micro-Command 101, it is necessary for W_c to be in the one-state upon entry into Phase IV. This is provided automatically, since $W_c P_{31}$ causes the transition from Phase I to Phase II, and no zero-set signals intervene thereafter.

The signals associated with the operation of the core storage are $M_m T_2'$ or $M_m T_1'$. Therefore, the one-state of M_m during Type IV operation leaves the core unaffected.

4.12 Description of Command Sequences:

The following describes the logical equations which represent computer operation during the execution of the various commands. With the exception of the Input and Halt Commands (009 and 012), all commands can be executed by either Type I or Type II orders. (009 and 012 are restricted to Type I orders.)

Previous sections have summarized the operation of the computer during Phases I and II (Part 4.7), and the B modification and memory search procedure* during Phase III (Part 4.8). The following comments are primarily concerned with operations which follow the memory search. When no such search is required, this will be noted.

Store A in Memory Position m: (002): AM m: When the memory search is successful, the machine switches to Phase IV. Phase IV is true for one word-interval. During this interval, the contents of the A Register are

---

* Recall that commands which require a memory search were included in the one-set logic for W_c. A successful search causes the computer to proceed from Phase III to Phase IV.

fed into M_{wn}, where M_{wn} represents the memory write flip-flop for channel n. Channel n is chosen by the Channel Select Register, C_6 through C_1. The channel selection logic, $f_n(C_6 - C_1)$, represents the output of either a matrix or logical gates, and is not shown in detail.

$$M_{wn}: \quad {}_1m_{wn} = f_n(C_6 - C_1)GU_1(002)A_0$$

$${}_0m_{wn} = f_n(C_6 - C_1)GU_1(002)A_0'$$

$$\text{End-of-Command Signal} = GU_1(002)P_{31}$$

Transfer the Contents of Memory Position n to the D Register: (003): MD n: When the memory search is successful, switch from Phase III to Phase IV. Phase IV is true for one word-interval. D_{31}, the D Register write flip-flop, receives the output of the selected memory channel.

$$D_{31}: \quad {}_1d_{31} = GU_1(003)W$$

$${}_0d_{31} = GU_1(003)W'$$

$$\text{End-of-Command Signal} = GU_1(003)P_{31}$$

The term W in the input to D_{31} represents, as before, the output of the memory channel selected by the Channel Select Register.

Transfer to the Short Loop: (004): MS n: The execution of this command causes the contents of eight words from the main drum memory to be transferred to the eight-word loop, Channel 0. The pick-up starts at address n, and automatically terminates eight words later.

To perform this command:

a. As before, W_c is used to search the drum during Phase III, and causes a transition to Phase IV when the proper address is found.

b. Flip-flops $N_4N_3N_2N_1$ monitor the transfer and terminate the process after eight words have been transferred. To accomplish this, flip-flops $N_4N_3N_2N_1$ are first set to the configuration 1110 during Phase III.

c. During Phase IV, when the actual transfer transpires, the four N flip-flops are decreased in count by one, at each P_{31}.

d. When $N_4'P_{31}$ is true, denoting that eight word-intervals have ensued, the process is terminated.

$$N_4: \quad {}_1n_4 = GU_1'(004)P_{31}$$
$$\quad {}_0n_4 = GU_1(004)N_3'N_2'N_1'P_{31}$$

$$N_3: \quad {}_1n_3 = GU_1'(004)P_{31}$$
$$\quad {}_0n_3 = GU_1(004)N_2'N_1'P_{31}$$

$$N_2: \quad {}_1n_2 = GU_1(004)N_2'N_1'P_{31} + GU_1'(004)P_{31}$$
$$\quad {}_0n_2 = GU_1(004)N_2N_1'P_{31}$$

$$N_1: \quad {}_1n_1 = GU_1(004)N_1'P_{31}$$
$$\quad {}_0n_1 = GU_1(004)N_1P_{31} + GU_1'(004)P_{31}$$

$$M_{WO}: \quad {}_1m_{WO} = GU_1(004)W$$
$$\quad {}_0m_{WO} = GU_1(004)W'$$

$$\text{End-of-Command Signal} = GU_1(004)N_4'P_{31}$$

Note that the higher stages of the N_4 - N_1 counter do not have the proper logic for a complete cycling-subtract counter. Since only the zero-state of N_4 is of significance, no harm results, and some logic is

eliminated.

Transfer from the Short Loop: (005): SM n: Command 005 causes the eight-word contents of the short loop, Channel 0, to be transferred to the main memory, starting at position n. The timing for this command is identical with that above; the only difference lies in the direction of transfer. The logical equations for flip-flops N_4 through N_1 will not be repeated; the only change required is to substitute the codes representing (004 + 005) in place of (004) alone.

$$M_{wn}: \quad 1^{m}_{wn} = f_n(C_6 - C_1)GU_1(005)M_{r0}$$

$$0^{m}_{wn} = f_n(C_6 - C_1)GU_1(005)M_{r0}'$$

$$\text{End-of-Command Signal} = GU_1(005)N_4'P_{31}$$

As before, M_{wn} represents the write flip-flop for Channel n, where Channel n is chosen by the Channel Select Register, C_6 through C_1. The term $f_n(C_6 - C_1)$ is again the output of a matrix or logical gates, and selects the particular channel to receive the output of the short loop channel.

Print: (006): PT n: The Print command causes the Flexowriter to print the symbol corresponding to the most significant six binary digits of the address accompanying this command. This command initiates the external typing operation, but once this is done, the computer is able to perform other commands even though the external typing operation has not been completed. However, should a Print operation be commanded before the Flexowriter has finished processing the previous output code, the computer will remain idle until the Flexowriter is capable of accepting the new output.

No memory search is required for this command. The computer normally remains in Phase III for one word-interval only. No operations occur during Phase III.

a. Phase IV is true for one word interval. During this time, the six most significant bits of the Address Register contents (in the address position) are shifted into the Input-Output Register, H_6 through H_1.

b. At the end of Phase IV, an interlock flip-flop J is set to the one-state. The use of the flip-flop is separately described below.

c. Also at the end of Phase IV, the End-of-Command signal causes the Flexowriter to type the symbol corresponding to the contents of the H Register.

The Input-Output Register, H_6 through H_1, is connected as a shift register, with H_6 receiving the output of the Address Register. Only the two end stages are shown here.

H_6: $\quad {}_1h_6 = GU_1(006)M_0P_{18-7}$

$\quad\quad {}_0h_6 = GU_1(006)M_0'P_{18-7} + IP_{31}$

H_1: $\quad {}_1h_1 = GU_1(006)H_2P_{18-7}$

$\quad\quad {}_0h_1 = GU_1(006)H_2'P_{18-7} + IP_{31}$

J: $\quad {}_1J = GU_1(006)P_{31}$

$\quad\quad {}_0J = IP_{31}$

End-of-Command Signal = $GU_1(006)P_{31}$

The interval during which the H flip-flops receive and shift

information is shown as P_{18-7}. The actual interval required is P_{18-13}, but since P_{18-7} is already available as a driver output, the extension of the interval does no harm.

The term IP_{31} indicates the completion of the external Flexowriter action and includes suitable delay terms to insure that the Flexowriter is indeed ready to receive further information. The logic for the I flip-flop is not described here; it is discussed in Part 4.6 and Part 4.14 (Fill State).

Once the Flexowriter operation has been initiated by the 006 Print command, the computer is free to perform any of the other commands or micro-commands, except that J remains on until the Flexowriter type operation is completed. Should another Print Command be called forth, the computer proceeds as follows:

a. The computer goes from Phase II to Phase III but remains in the Phase III condition as long as J remains on.

b. When the Flexowriter finishes its operation, J is zero-set by IP_{31}, enabling the computer to proceed to Phase IV. Thereafter, the operation is as described above.

$$U_2: \quad {}_1u_2 = U_2'U_1P_{31}$$

$$U_1: \quad {}_1u_1 = GU_1'(006)J'P_{31}$$

$$\quad {}_0u_1 = (T_2' + T_1')U_2'U_1P_{31}$$

$$J: \quad {}_0J = IP_{31}$$

$$\text{End-of-Command Signal} = GU_1(006)P_{31}$$

Partial Substitute: (007): PS m: The execution of the Partial Substitute command causes the bits in the address section of the word in the A Register to be substituted into the corresponding section of the word in position m of the memory. The most direct way to accomplish this is to conduct a memory search for position m, and then to transfer the A Register contents into m for the interval defining the address section.

If there are objections to writing on the memory for partial word intervals, an alternate method would be to pick-up the selected word from memory, superimpose it onto the existing A Register contents, and then re-write the modified word back into the memory. The re-writing of information into the memory is easily accomplished by returning to Phase III and changing the configuration of the Q Register, which stores the command code, so that the computer performs command 002, AM m.

The sequence below describes the former, simpler method, since this appears to be satisfactory.

The computer proceeds through a search of the memory during Phase III, as previously described. When the search is successful, the computer goes from Phase III to Phase IV. The machine remains in Phase IV for one word-interval only.

During Phase IV, the contents of the A Register during P_{18-7} are written into the selected memory word position.

$$M_{wm}: \quad {}_1M_{wm} = f_n(C_6 - C_1)GU_1(007)A_0P_{18-7}$$

$${}_0M_{wm} = f_n(C_6 - C_1)GU_1(007)A_0'P_{18-7}$$

$$\text{End-of-Command Signal} = GU_1(007)P_{31}$$

Extract: (008): EXT m: The Extract command causes the contents of the

A Register to be erased, except where there are one-bits present in the contents of position m in the memory. After a search during Phase III for the word in position m, the computer switches to Phase IV. During Phase IV, the contents of the A Register are modified in the manner shown below.

$$A_{31}: \quad {}_1a_{31} = GU_1(008)WA_0$$

$${}_0a_{31} = GU_1(008)(W' + A_0')$$

$$\text{End-of-Command Signal} = GU_1(008)P_{31}$$

Input: (009): IN: The Input command causes the computer to revert to the Idle State ($V_2'V_1'$) so that computation ceases, and also supplies an external signal which starts the punched-tape reader.

a. The computer remains in Phase III for one word interval, and then reverts to Phase I.

b. At the end of Phase III, V_2 and V_1 are zero-set to the Idle State ($V_2'V_1'$).

c. The End-of-Command signal, $GT_1'(009)P_{31}$, is also the signal to start the punched-tape reader.

$$V_2: \quad {}_0v_2 = GT_1'(009)P_{31}$$

$$V_1: \quad {}_0v_1 = GT_1'(009)P_{31}$$

$$U_2: \quad {}_0u_2 = GT_1'(009)P_{31}$$

$$\text{End-of-Command Signal} = GT_1'(009)P_{31}$$

The Input command cannot be executed from the core memory, since the code for 009 is multiplied by T_1', limiting operation to Type I. This insures that the machine cannot revert to the Idle State when executing a Type II order unless the Return-to-Main-Control micro-command is used.

Store the Augmented Order Counter Contents: (010): OCM m: Command 010 causes the contents of the Order Counter to be increased by one and then stored in the address portion of the word in position m of the memory. Since the Order Counter contains the address of the order normally next to be executed, command 010 allows the programmer to store the address of the order to be executed two orders hence. To insert a sub-routine into a program, therefore, the sequence of orders 010 m_1 and 215 m_2 (Type IV) would be used, where m_1 is the address of the last order in the sub-routine, and m_2 is the address of the beginning of the sub-routine. The final order in the sub-routine would be 215 m_3 (Type IV), with the address m_3 having been specified by the execution of command 010 (therefore, m_3 = Order Counter Contents + 1).

The execution of the 010 command is very similar to that for the Partial Substitute command (007), except that the information written into the address portion of the word in position m is obtained from logic which adds one to the Order Counter contents. Here again, if there are objections to the partial-word-interval writing operation, the contents of m can be placed into the A Register, modified, and then written back into position m by generating command 002. The former method is described below.

a. At the completion of a successful search, when the computer

switches to Phase IV, O_c is set to the one-state.

b. In Phase IV, during the address interval of the word, the memory write flip-flop, M_{wm}, receives the output of the Order Counter O_O, and the setting of O_c, such that the increase-by-one information is stored.

$$O_c: \quad {}_1O_c = GU_1'(010)W_cP_{31}$$

$$\quad {}_0O_c = GU_1(010)O_cO_O'P_{18\text{-}7}$$

$$M_{wm}: \quad {}_1M_{wm} = f_n(C_6 - C_1)GU_1(010)(O_cO_O' + O_c'O_O)P_{18\text{-}7}$$

$$\quad {}_0M_{wm} = f_n(C_6 - C_1)GU_1(010)(O_cO_O + O_c'O_O')P_{18\text{-}7}$$

End-of-Command Signal = $GU_1(010)P_{31}$

Transfer the Address in Position m into the B Register: (011): MB m: This command causes the address portion of the word in position m to be placed into the B Register. After the memory search during Phase III, the computer substitutes the required information into the B Register during the one word interval when Phase IV is true.

$$B_{31}: \quad {}_1B_{31} = GU_1(011)WP_{18\text{-}7}$$

$$\quad {}_0B_{31} = GU_1(011)W'P_{18\text{-}7}$$

End-of-Command Signal = $GU_1(011)P_{31}$

Halt: (012): H: The Halt Command causes the machine to assume the Idle State, and is executed in exactly the same manner as the Input Command 009, except that no signal is supplied to the punched-tape reader. A Halt Command cannot be executed from the core memory.

Decrease B (Conditional Command): (013): DB m: Command 013 causes the contents of the B Register to be decreased by one. If B is zero at the start of this command, and hence becomes negative as a result of the decrease, the computer proceeds to the next order as specified by the Order Counter. If the resulting B is either zero or positive, the computer proceeds to the order specified by the address m.

The order is executed as follows:

a. The computer remains in Phase III for one word interval only, and then proceeds to Phase IV.

b. Flip-flop O_c is one-set at the end of Phase III, so that it has an initial one-value at the beginning of Phase IV.

c. During the first word interval in Phase IV, the contents of B are decreased by one, under command of O_c.

d. At the end of the word interval, the final setting of O_c indicates which of the two cases above is satisfied by B.

e. If O_c is still in the one-state at the end of the first word interval of Phase IV, this indicates that the resulting B value is negative, and causes the computer to switch to Phase I; hence, the next order is specified by the existing contents of the Order Counter in the usual fashion.

f. If O_c is in the zero-state at the end of the first word interval of Phase IV, then W_c is set to the one-state. (W_c is zero-set during Phase II and cannot since have been one-set.) This causes the contents of the Address Register to be transferred into the Order Counter.

g. After the second word interval of Phase IV, the computer returns to Phase I. Since the Order Counter now contains the address m which was in the Address Register, this determines the location of the

next order to be executed.

$$^{0}_{c}: \quad {}_{1}{}^{o}_{c} = GU_{1}'(013)P_{31}$$

$${}_{0}{}^{o}_{c} = GU_{1}(013)B_{0}{}^{0}_{c}P_{18-7}$$

$$B_{31}: \quad {}_{1}{}^{b}_{31} = GU_{1}(013)({}^{0}_{c}B_{0}' + {}^{0}_{c}'B_{0})P_{18-7}$$

$${}_{0}{}^{b}_{31} = GU_{1}(013)({}^{0}_{c}B_{0} + {}^{0}_{c}'B_{0}')P_{18-7}$$

$$W_{c}: \quad {}_{1}{}^{w}_{c} = GU_{1}(013)W_{c}'{}^{0}_{c}'P_{31}$$

$${}_{0}{}^{w}_{c} = GU_{1}(013)W_{c}P_{31} + (T_{2}' + T_{1}')U_{2}'U_{1}P_{31}$$

$$^{0}_{31}: \quad {}_{1}{}^{o}_{31} = GU_{1}(013)W_{c}M_{0}P_{18-7}$$

$${}_{0}{}^{o}_{31} = GU_{1}(013)W_{c}M_{0}'P_{18-7}$$

$$\text{End-of-Command Signal} = GU_{1}(013)({}^{0}_{c} + W_{c})P_{31}$$

Transfer the Contents of B into the A Register: (014): BA: To execute this command, the computer again "idles" in Phase III for a word interval, and then proceeds to Phase IV. Phase IV is true for one word interval, during which the contents of B are transferred into the A Register.

$$A_{31}: \quad {}_{1}{}^{a}_{31} = GU_{1}(014)B_{0}$$

$${}_{0}{}^{a}_{31} = GU_{1}(014)B_{0}'$$

$$\text{End-of-Command Signal} = GU_{1}(014)P_{31}$$

Increase the Contents of B by One: (015): IB: Again, the computer performs no operations during the word-interval when Phase III is true. At

the transition into Phase IV, flip-flop 0_c is set to the one-state and is used to command the B + 1 operation during Phase IV.

$$0_c: \quad 1^0{}_c = GU_1'(015)P_{31}$$

$$0^0{}_c = GU_1(015)0_cB_0'P_{18-7}$$

$$B_{31}: \quad 1^b{}_{31} = GU_1(015)(0_cB_0' + 0_c'B_0)P_{18-7}$$

$$0^b{}_{31} = GU_1(015)(0_cB_0 + 0_c'B_0')P_{18-7}$$

$$\text{End-of-Command Signal} = GU_1(015)P_{31}$$

(Note that the logic for B_{31} is identical for commands 013 and 015, but that the zero-set terms for 0_c differ.)

4.13 Description of Micro-Command Execution Sequences:

This section describes the sequences involved in executing the micro-commands which comprise a micro-order. In all cases, micro-commands are executed during Phase IV, and never require more than one word interval for completion.

Figure 3 illustrates the configuration of a micro-order obtained from the core memory. The various component micro-commands are described below, progressing from left to right in Figure 3.

N Counter Code:

A code bit in the msd of a micro-order causes the contents of the N Counter (N_5 through N_1) to be decreased by one count. If the resulting N Counter contents are positive or zero, the machine proceeds next to execute the micro-order in the row position specified by the Y_n Address

portion of the micro-order. However, if the resulting N Counter value is negative (i.e., if N were equal to zero before the decrease by one), the micro-order next in the core is executed. In either case, the machine will execute the micro-commands which accompany the N code bit.

The sequence is as follows:

a. If X_{27} is true at P_1, this indicates that a code bit was obtained at the previous P_{31}. The contents of the N Counter are decreased by one.

b. If the N Counter has the configuration $N_5'N_4'N_3'N_2'N_1'$ at P_0 (prior to the decrease by one), then the N Counter contents will become negative when decreased by one. Any configuration except the zero-state of N will one-set X_{11} at P_1.

c. Later descriptions of the operation of the Y_n Control Code show that the one-state of X_{11} at P_{28} causes the Y_n Address portion of the micro-order to be transferred into the Row Select Register, Y_7 through Y_1. Therefore, if N is not zero prior to the decrease by one, the next micro-order will be obtained from the position specified by the Y_n Address. If N is zero, the micro-order in the following core position will be executed.

Only the end stages of the N Counter, N_5 and N_1, are shown below.

$$N_5: \quad {}_1n_5 = G_m X_{27} N_5' N_4' N_3' N_2' N_1' P_1$$

$${}_0n_5 = G_m X_{27} N_5 N_4' N_3' N_2' N_1' P_1$$

$$N_1: \quad {}_1n_1 = G_m X_{27} N_1' P_1$$

$${}_0n_1 = G_m X_{27} N_1 P_1$$

$$X_{11}: \quad {}_1X_{11} = G_m X_{27}(N_5 + N_4 + N_3 + N_2 + N_1)P_1$$

Arithmetic Micro-Command Execution:

The following are descriptions of the logical operations involved in executing the Arithmetic Micro-Commands.

Complement A: (101): comp. A: This micro-command causes the contents of the A Register to be complemented, i.e., the true binary complement is generated and written into the A Register.

The complementation process requires that W_c be in the one-state at the beginning of the word interval during which the A Register contents are modified. The logic indicated below insures that W_c is indeed in the one-state both at entry into and during the Micro-Mode.

$$W_c: \quad {}_1W_c = M_m T_1 X_{13}' P_{31}$$

$${}_0W_c = G_m X_{13} P_{31} + G_m(101)A_0 P_{31}'$$

$$A_{31}: \quad {}_1a_{31} = G_m(101)(W_c A_0 + W_c' A_0')$$

$${}_0a_{31} = G_m(101)(W_c A_0' + W_c' A_0)$$

The one-set term for W_c turns W_c on upon entry into the Micro-Mode for a Type II order, whether the entry is for the first time in the execution of the order, or whether it is because of a digression to execute a command from core. The same term continues to set W_c on during the Micro-Mode interval. When X_{13} is true, indicating that a command is to be executed from core, W_c is zero-set to prevent interference with the command sequence. For a Type IV order, no one-set for W_c is needed, since W_c is one-set upon leaving Phase I, and no zero-sets intervene.

During the actual complementation cycle, the first one-bit in the A

Register sets W_c to the zero-state. Analysis will indicate that the logic shown for A_{31} complements the A Register contents.

A + D into A: (102): a ADA: This micro-command causes the sum of the A and D Registers to be written back into the A Register. The logic for this micro-command would normally be very simple, but is complicated by the possibility of simultaneous micro-commands in the Control Micro-Command section, which can call for the shifting of the A Register contents either to the right or to the left.

The logic shown below illustrates the operation of micro-command 102, both with and without simultaneous shift micro-commands.

$$A_s = G_m(202 + 206)'(A_0'D_0'K + A_0'D_0K' + A_0D_0'K' + A_0D_0K)$$
$$+ G_m(202 + 206)(A_1'D_0'K + A_1'D_0K' + A_1D_0'K' + A_1D_0K)$$

$$A_s' = G_m(202 + 206)'(A_0D_0K' + A_0D_0'K + A_0'D_0K + A_0'D_0'K')$$
$$+ G_m(202 + 206)(A_1D_0K' + A_1D_0'K + A_1'D_0K + A_1'D_0'K')$$

K: $$_1k = G_m(102)(202 + 206)'A_0D_0P_{31}'$$
$$+ G_m(102)(202 + 206)A_1D_0P_{31}'$$

$$_0k = G_m(102)(202 + 206)'A_0'D_0'$$
$$+ G_m(102)(202 + 206)A_1'D_0' + P_{31}$$

A_{31}: $$_1a_{31} = G_m(102)(203 + 205 + 207)'A_s$$
$$+ G_m(114P_{31})'(106)'(203 + 205 + 207)C_1P_0'$$

$$_0a_{31} = G_m(102)(203 + 205 + 207)'A_s'$$
$$+ G_m(115P_{31})'(203 + 205 + 207)C_1'P_0'$$

C_1: $\quad {}_1c_1 = G_m(102)A_s$

$\quad\quad {}_0c_1 = G_m(102)A_s$'

The comments below are focused primarily on the addition operation; the effects of the various shift micro-commands are mentioned as necessary, but further details of their operation are contained in those later sections devoted to the shift micro-commands.

Consider the logic for the A Sum Driver, A_s. In the absence of shift-right micro-commands, as denoted by (202 + 206)', A_s represents the direct sum of the A and D Registers, since A_0 and D_0 are added using the carry flip-flop K. However, should a shift-right of the A Register be required, as shown by (202 + 206), then A_1 and D_0 are added; therefore, D is added to the shifted A value, with this resultant sum then written into the A Register.

The inputs to the K flip-flop reflect the above operations. K is zero-set by P_{31}, so that it is zero at the start of the addition interval.

For the two cases of shift-right or recirculate (these can also be summarized as non-shift-left), A_s directly determines the input to A_{31}. Therefore, (203 + 205 + 207)', which represents the non-shift-left case, causes A_{31} to be set by A_s.

There remains the possibility of combining addition with shift-left operation. This is accomplished by using C_1 on a time-shared basis as a delay flip-flop to enable a shift-left of the A Register contents. (The function of C_1 as part of the Channel Select Register is not required during Micro-Mode; hence it performs the function of A_{32} described in Part 4.4 of this thesis.)

If the Addition Micro-Command 102 occurs, C_1 receives the A_s driver output. If, at the same time, any of the shift-left micro-commands 203, 205 or 207 occur, C_1, which represents the one-clock-delayed A_s value, is fed into the A_{31} flip-flop. (When a shift-left operation is commanded, A_s must represent the sum of A_0 and D_0, since a shift-right command could not simultaneously occur.) Therefore, a combined addition and shift-left operation causes the sum of A and D to be left-shifted when written into the A Register. It will be recalled that a combined addition and shift-right operation caused the sum of D and the shifted A to be placed into the A Register.

The terms $(114P_{31})'(106)'$ in the one-set logic for A_{31}, and the term $(115P_{31})'$ in the zero-set, are explained later in connection with those particular micro-commands.

A - D into A: (103): s ADA: The execution of this micro-command is identical to that for 102, except that the D Register contents are subtracted from those of the A Register. Except for the change in the logic for the carry flip-flop K, as noted below, all other logic terms are the same as for 102. Therefore, except for K, the flip-flops shown in the discussion for 102 should have 102 + 103 substituted for 102.

$$K: \quad {}_1k = G_m(103)(202 + 206)'A_0'D_0P_{31}'$$
$$+ G_m(103)(202 + 206)A_1'D_0P_{31}'$$
$${}_0k = G_m(103)(202 + 206)'A_0D_0'$$
$$+ G_m(103)(202 + 206)A_1D_0' + P_{31}$$

A + D into R: (104): e ADR: In executing this micro-command, the sum of A and D are written into the R Register, while the contents of A and D are normally left undisturbed. R_{31}, the write flip-flop for the R

Channel, receives the output of the A_s Driver. A simultaneous shift micro-command in the Control Micro-Command portion of the micro-order affects the execution of 104 only in the manner in which it affects A_s. Hence, a simultaneous shift-right of the A Register causes the sum of D and the shifted A to be written into R; a simultaneous shift left of the A Register has no affect on the sum placed into R.

$$R_{31}: \quad {}_1r_{31} = G_m(104)A_s$$

$${}_0r_{31} = G_m(104)A_s'$$

$$K: \quad {}_1k = G_m(202 + 206)'(104)A_0D_0P_{31}'$$

$$+ G_m(202 + 206)(104)A_1D_0P_{31}'$$

$${}_0k = G_m(202 + 206)'(104)A_0'D_0'$$

$$+ G_m(202 + 206)(104)A_1'D_0' + P_{31}$$

The logic for K, the Carry flip-flop, is identical to that used for Micro-Command 102.

A - D into R: (105): s ADR: The D Register contents are subtracted from the A Register and placed into R. Comments which apply to 104 also apply for this micro-command. The logic for R_{31} is identical to that of 104; therefore, substitute 104 + 105 for 104 in the input to R_{31}.

The carry flip-flop K operates as in 103; therefore, substitute 103 + 105 for 103, in the input to K.

Clear A: (106): cl A: In executing the Clear A micro-command, the write flip-flop for the A Register, A_{31}, is zero-set at P_0. None of the one-set terms to A_{31} is allowed to be true during the entire word interval. The initial setting of A_{31} propagates through the A Register, but

is erased at the end of the word interval.

$$A_{31}: \quad {}_1a_{31} = G_m(106)\text{'(All possible one-set terms)}$$

$${}_0a_{31} = G_m(106)P_0$$

Because of the manner in which the logic is written, the Clear A operation takes precedence over any other simultaneous operation which may attempt to write information into the A Register.

Clear R: (107): cl R: The Clear R micro-command operates as above. R_{31} is zero-set at P_0, and all one-set terms are rendered inoperative.

$$R_{31}: \quad {}_1r_{31} = G_m(107)\text{'(All possible one-set terms)}$$

$${}_0r_{31} = G_m(107)P_0$$

Clear D: (108): cl D: The clear D micro-command causes D_{31} to be zero-set at P_0, and negates all one-set terms.

$$D_{31}: \quad {}_1d_{31} = G_m(108)\text{'(All possible one-set terms)}$$

$${}_0d_{31} = G_m(108)P_0$$

0 into msd D: (109): 0 msd D: This micro-command causes a zero to be placed into the most significant digit position of the D Register. This is accomplished by allowing D to recirculate normally, except at P_{31} when D_{31} is set to zero. (The logic for recirculation is not shown here.) This micro-command overrides any attempts to write a one-digit into the msd of D.

$$D_{31}: \quad {}_1d_{31} = (109P_{31})\text{'(All possible one-set terms)}$$

$${}_0d_{31} = G_m(109)P_{31}$$

0 into lsd R: (110): 0 lsd R: In executing this micro-command, R_0 is zero-set at P_{31}. This places a zero-digit in the least significant position of the R Register.

$$R_0: \quad {}_1r_0 = G_m(110P_{31})'\text{(All possible one-set terms)}$$

$${}_0r_0 = G_m(110)P_{31}$$

1 into lsd R: (111): 1 lsd R: The procedure here is similar to that for 110, except that the R_0 flip-flop is one-set at P_{31}.

$$R_0: \quad {}_1r_0 = G_m(111)P_{31}$$

$${}_0r_0 = G_m(111P_{31})'\text{(All possible zero-set terms)}$$

0 into msd R: (112): 0 msd R: The logic for this micro-command is similar to that for 109, but is applied to the R channel. R_{31} is zero-set at P_{31}.

$$R_{31}: \quad {}_1r_{31} = G_m(112P_{31})'\text{(All possible one-set terms)}$$

$${}_0r_{31} = G_m(112)P_{31}$$

1 into msd R: (113): 1 msd R: At P_{31}, flip-flop R_{31} is one-set. At all other times, the contents of the R Register are recirculated.

$$R_{31}: \quad {}_1r_{31} = G_m(113)P_{31}$$

$${}_0r_{31} = G_m(113P_{31})'\text{(All possible zero-set terms)}$$

0 into msd A: (114): 0 msd A: At P_{31}, flip-flop A_{31} is set to the zero-state. At all other times, the A Register contents are recirculated.

A_{31}: $_1a_{31} = G_m(114P_{31})'$(All possible one-set terms)

$_0a_{31} = G_m(114)P_{31}$

1 into msd A: (115): 1 msd A: As above, except that A_{31} is one-set at P_{31}.

A_{31}: $_1a_{31} = G_m(115)P_{31}$

$_0a_{31} = G_m(115P_{31})'$(All possible zero-set terms)

Control Micro-Command Execution:

The following section describes the logical equations involved in executing the Control Micro-Commands.

Return to Main Control: (201): rmc: This micro-command causes the computer to return to Phase I after executing the micro-order in which this micro-command is placed. The details of this operation have already been described in Part 4.9, describing the return to Phase I for a Type II order.

Shift A Right: (202): sr A: This micro-command causes the contents of the A Register to be shifted to the right by one position. A zero-bit is inserted at the most significant digit position; the least significant digit is lost. If an addition or subtraction is commanded simultaneously, the shifted A value is used in the arithmetic operation.

Most of the logic below has already been discussed in connection with add and subtract micro-commands. In the absence of add or subtract codes, as denoted by $(102 + 103)'$, A_1 is written back into the A Register, causing a shift-right of the A contents. When $(102 + 103)$ is true, A_s, which generates the sum or difference, is sensed by A_{31}. The

effect of the shift commands is included in the logic for A_s.

Note in particular, that A_{31} is zero-set at P_{31}, causing the msd of A to be zero. However, a simultaneous micro-command 115, which causes the msd of A to be set to one, would have priority.

$$A_{31}: \quad {}_1a_{31} = G_m(102 + 103)'(106)'(202)A_1P_{31}'$$
$$+ G_m(102 + 103)(203 + 205 + 207)'A_s$$
$${}_0a_{31} = G_m(102 + 103)'(202)A_1'P_{31}'$$
$$+ G_m(102 + 103)(203 + 205 + 207)'A_s'$$
$$+ G_m(115)'(202)P_{31}$$

Term P_{31}' in the zero-set is redundant, but is useful in combining with the logic for 204.

Shift A Left: (203): sl A: The normal operation of this micro-command causes the contents of A to be shifted left by one position, with the resultant loss of the msd and with a zero-bit inserted into the lsd position. When combined with an add or subtract micro-command, the resultant sum or difference is written into the A Register, displaced by one position to the left, with the same effect on the end bits as discussed above.

The shift-left effect is obtained by feeding either A_0 or A_s into the delay flip-flop C_1. At all times other than P_0, the contents of C_1 are fed into A_{31}. At P_0, however, A_{31} is zero-set, causing the lsd of A to contain a zero-bit after the shift operation is completed. (Another possible method would be to set A_0 to the zero-state at P_{31}. Conceptually, this is simpler; however, the logic required for executing all of the shift-left variations, micro-commands 203, 205, and 207,

appears to be simpler if the present approach is used.)

$$C_1: \quad {}_1c_1 = G_m(102 + 103)A_s + G_m(102 + 103)'A_0$$

$${}_0c_1 = G_m(102 + 103)A_s' + G_m(102 + 103)'A_0'$$

$$A_{31}: \quad {}_1a_{31} = G_m(114P_{31})'(106)'(203)C_1P_0'$$

$${}_0a_{31} = G_m(115P_{31})'(203)C_1'P_0' + G_m(203)P_0$$

Term P_0' is here redundant in the zero-set for A_{31}, but is necessary for the proper operation of 205.

Circular Shift A Right: (204): src A: This micro-command causes the contents of the A Register to be shifted right by one position, but with the original lsd of A written back into the msd position. Hence, none of the original information is lost.

Except at P_{31}, A_{31} receives the output of A_1 as before, resulting in a shift-right operation. At P_{31}, flip-flop A_{31} is set to the value stored in A_0. Since A_0 is not required during a shift right operation, it is "frozen" for the duration of the word interval, and thus stores the original lsd existing at the beginning of the shift operation. The recirculation terms which cause continuous transfer of information from A_1 to A_0 are negated during the execution of this micro-command. Since recirculation logic is not normally shown in these descriptions, the inhibition of transfer from A_1 to A_0 is not apparent below.

$$A_{31}: \quad {}_1a_{31} = G_m(102 + 103)'(106)'(204)A_1P_{31}'$$

$$+ G_m(106)'(114)'(204)A_0P_{31}$$

$${}_0a_{31} = G_m(102 + 103)'(204)A_1'P_{31}'$$

$$+ G_m(115)'(204)A_0'P_{31}$$

$$A_0: \quad {}_1a_0 = G_m(204)A_1P_{31} + \text{Non-Circulate}$$

$${}_0a_0 = G_m(204)A_1'P_{31} + \text{Non-Circulate}$$

Circular Shift A Left: (205): slc A: This micro-command causes a one-shift to the left, but with the original msd of A written into the vacated lsd position. The basic shift-left is caused by using C_1 again as a delay flip-flop. The substitution of the original msd into the lsd position is effected by causing A_{31} to receive no signal at P_0. This delays the original A_{31} bit so that it automatically fills the vacancy caused by the shift-left operation.

$$A_{31}: \quad {}_1a_{31} = G_m(114P_{31})'(206)'(205)C_1P_0'$$

$${}_0a_{31} = G_m(115P_{31})'(205)C_1'P_0'$$

$$C_1: \quad {}_1c_1 = G_m(102 + 103)'A_0$$

$${}_0c_1 = G_m(102 + 103)'A_0'$$

Shift A and R Right: (206): sr AR: Both the A and R Registers are right-shifted, with the original lsd of A placed into the msd position of R, and with a zero-bit inserted into the msd position of A. The operation of the A Register is identical to that for 202; A_{31} receives the A_1 flip-flop values except at P_{31}, when it is zero-set to insert the zero-bit into the msd position. A_0 stores the original lsd of the A Register in the same manner as in 204.

The R Register is simultaneously shifted right, by allowing R_{31} to receive the contents of R_1. At P_{31}, R_{31} is set to the state represented by A_0, thus inserting the lsd of A into the msd position of R.

$$A_{31}: \quad {}_1a_{31} = G_m(102 + 103)'(106)'(206)A_1P_{31}'$$

$${}_0a_{31} = G_m(102 + 103)'(206)A_1'P_{31}'$$

$$+ G_m(115)'(206)P_{31}$$

$$A_0: \quad {}_1a_0 = G_m(206)A_1P_{31} + \text{Non-Circulate}$$

$${}_0a_0 = G_m(206)A_1'P_{31} + \text{Non-Circulate}$$

$$R_{31}: \quad {}_1r_{31} = G_m(107)'(206)[R_1P_{31}' + (112)'A_0P_{31}]$$

$${}_0r_{31} = G_m(206)[R_1'P_{31}' + (113)'A_0'P_{31}]$$

Shift A and R Left: (207): sl AR: Both the A and R Registers are left-shifted. The original msd of R is placed into the lsd position of A; the lsd position of R has a zero-bit inserted. The A Register is left-shifted as before, by interposing flip-flop C_1 as an added delay in the A loop. A_{31} receives the C_1 output except at P_0; at P_0, the contents of R_{31} are used to set A_{31}.

The R Register is left-shifted in a similar manner, by using C_2 as the delay in the R loop. R_{31} receives C_2 at all times except P_0; at P_0, R_{31} is zero-set to insert the zero in the lsd position.

$$A_{31}: \quad {}_1a_{31} = G_m(114P_{31})'(106)'(207)C_1P_0'$$

$$+ G_m(106)'(207)R_{31}P_0$$

$${}_0a_{31} = G_m(115P_{31})'(207)C_1'P_0' + G_m(207)R_{31}'P_0$$

$$C_1: \quad {}_1c_1 = G_m(102 + 103)'A_0$$

$${}_0c_1 = G_m(102 + 103)'A_0'$$

$$R_{31}: \quad 1^{r}_{31} = G_m(107)'(112P_{31})'(207)C_2P_0'$$

$$0^{r}_{31} = G_m(113P_{31})'(207)(C_2' + P_0)$$

$$C_2: \quad 1^{c}_{2} = G_m(207)R_0$$

$$0^{c}_{2} = G_m(207)R_0'$$

Transfer A into the Address Register M: (208): tr AMr: This micro-command causes the address portion of the word in the A Register to be transferred into the M or Address Register. M_{31} receives A_0 during the P_{18-7} interval defining the address interval.

$$M_{31}: \quad 1^{m}_{31} = G_m(208)A_0P_{18-7}$$

$$0^{m}_{31} = G_m(208)A_0'P_{18-7}$$

Transfer the Contents of the A Register into the R Register: (209): tr AR: The contents of the A Register are transferred directly into the R Register; the A Register is recirculated and remains unchanged.

$$R_{31}: \quad 1^{r}_{31} = G_m(107)'(112P_{31})'(209)A_0$$

$$0^{r}_{31} = G_m(113P_{31})'(209)A_0'$$

Transfer the Contents of the R Register into the A Register: (210): tr RA: This micro-command is the opposite of 209, and causes R to be placed into A, meanwhile preserving the R Register contents.

$$A_{31}: \quad 1^{a}_{31} = G_m(114P_{31})'(106)'(210)R_0$$

$$0^{a}_{31} = G_m(115P_{31})'(210)R_0'$$

Exchange the Contents of the A and R Registers: (211): ex AR: Essentially, this combines the effects of 209 and 210. The write flip-flops, A_{31} and R_{31}, receive the outputs of the opposing registers.

$$A_{31}: \quad 1^{a}_{31} = G_m(114P_{31})'(106)'(211)R_0$$
$$0^{a}_{31} = G_m(115P_{31})'(211)R_0'$$

$$R_{31}: \quad 1^{r}_{31} = G_m(107)'(112P_{31})'(211)A_0$$
$$0^{r}_{31} = G_m(113P_{31})'(211)A_0'$$

Exchange the Contents of the A and D Registers: (212): ex AD: This is similar to the above, but involves the A and D Registers.

$$A_{31}: \quad 1^{a}_{31} = G_m(114P_{31})'(106)'(212)D_0$$
$$0^{a}_{31} = G_m(115P_{31})'(212)D_0'$$

$$D_{31}: \quad 1^{d}_{31} = G_m(108)'(109P_{31})'(212)A_0$$
$$0^{d}_{31} = G_m(212)A_0'$$

Exchange the Contents of the R and D Registers: (213): ex RD: Again, the operation is as above, but with the remaining combination of the R and D Registers.

$$R_{31}: \quad 1^{r}_{31} = G_m(107)'(112P_{31})'(213)D_0$$
$$0^{r}_{31} = G_m(113P_{31})'(213)D_0'$$

$$D_{31}: \quad 1^{d}_{31} = G_m(108)'(109P_{31})'(213)R_0$$
$$0^{d}_{31} = G_m(213)R_0'$$

Transfer Part of the D Register into the N Counter: (214): tr DN:

This micro-command causes five bits of the D Register, in positions P_6 through P_2, to be transferred to stages N_5 through N_1 of the Operation Code Counter (or, N Counter). Flip-flops N_5 through N_1 operate as a shift register during the interval P_{6-0}, with the output of the D Register, D_0, fed into N_5 and progressively shifted to the right. Only the end stages, N_5 and N_1, are shown below.

$$N_5: \quad {}_1n_5 = G_m(214)D_0P_{6-0}$$

$${}_0n_5 = G_m(214)D_0{}'P_{6-0}$$

$$N_1: \quad {}_1n_1 = G_m(214)N_2P_{6-0}$$

$${}_0n_1 = G_m(214)N_2{}'P_{6-0}$$

Transfer the Address Register to the Order Counter: (215): tr MrOC:

This micro-command causes the address bits stored in the Address Register to be transferred to the Order Counter. The transfer is effected during the P_{18-7} interval.

$$O_{31}: \quad {}_1O_{31} = G_m(215)M_0P_{18-7}$$

$${}_0O_{31} = G_m(215)M_0{}'P_{18-7}$$

Control Input Codes and Control Selector Codes:

The Control Flip-Flops, F_0 through F_3, are used to control the conditional branching of the micro-routines. These flip-flops can be set by one of several conditions. The desired input quantity is chosen by using the Control Input Code (300-307), while the particular flip-flop to be set by the selected input is chosen by the Control Selector Code

(400-403).

The list of possible input terms, and the particular Control Input Codes which select among them, are summarized below:

301. lsd R
302. lsd A
303. msd A
304. msd D
305. Carry at P_{31}
306. Set one
307. Set zero

The selected term is used to set one of the flip-flops F_0 through F_3, as determined by the 400 through 403 codes, respectively.

The logic for Control Flip-Flop F_0 is shown below. The equations for the remaining Control Flip-Flops are identical, except for the substitution of the other 400 series code configurations.

$$F_0: \quad {}_1f_0 = G_m(400)[(301)R_0 + (302)A_0 + (303)A_{31} + (304)D_{31} + (306)]P_0 + G_m(400)(305)KP_{31}$$

$${}_0f_0 = G_m(400)[(301)R_0' + (302)A_0' + (303)A_{31}' + (304)D_{31}' + (307)]P_0 + G_m(400)(305)K'P_{31}$$

Except for 305 (and 306 and 307, which are unconditional), the P_0 sample time causes the Control Flip-Flops to be set to the state of the selected variable existing at the start of the word interval in which the code is used. The 305 code, which senses an overflow of the Carry flip-flop at P_{31}, selects a condition occurring at the <u>end</u> of the word interval in which the code appears. (See previous comments in Part 3.6.)

<u>Y_n Control Code: (500 series):</u>

The three bits which represent the Y_n Control Code are used, in

general, to control the branching of the micro-routines. The applicable conditions are described below.

a. Codes 501 through 504 select one of the four Control Flip-Flops. If the selected Control Flip-Flop is in the one-state, the computer proceeds to the position in the core memory specified by the Y_n Address bits in the micro-order. If the selected Control Flip-Flop is in the zero-state, the computer proceeds to the micro-order next in the core.

b. Code 505 causes an unconditional transfer to the micro-order specified by the Y_n address.

c. Code 506 does not affect the course of the micro-routine, but causes the five lsd bits of the Y_n Address to be placed into the N Counter.

d. Code 507 denotes that the following word from the core memory is to be interpreted as a command, rather than as a combination of micro-commands. The computer will then follow the sequences described in Part 4.9, Command Execution During Type II.

The conditions described above basically determine whether the Row Select Register, Y_7 through Y_1, is to advance by one count, or whether it is to receive the contents of the Y_n Address portion of the micro-order. The three bits of the Y_n Control Code are read out at P_{31} into flip-flops X_{11}, X_{12}, and X_{13}. From their own configuration and that of the Control Flip-Flops, the X_{11}, X_{12}, and X_{13} flip-flops are then set so that at P_{28}, when the Row Select Register Y_7 through Y_1 normally advances, only one of the three flip-flops is in the one-state. Hence, if, at P_{28}:

a. X_{11} is in the one-state, transfer the Y_n Address into Y_7 - Y_1.

b. X_{12} is in the one-state, place the five least significant bits

of the Y_n Address into the N Counter.

c. X_{13} is in the one-state, interpret the following core output as a Non-Micro-Order.

The re-set time for flip-flops X_{11} - X_{13} could be chosen for any time between P_0 and P_{28}. P_1 was selected. Because of the relative sampling times involved, a Control Flip-Flop which is set by any input except 305 can be chosen by a 500 code contained in the same micro-order. An input selected by 305, however, requires that the Control Flip-Flop be sampled in a later word interval.

The logical equations for X_{11} - X_{13} are shown below:

$$X_{13}: \quad {}_1X_{13} = M_m T_2' z_{13} P_{31} + \boxed{(507)P_1}$$

$${}_0X_{13} = M_m T_2' z_{13}' P_{31} + G_m(507)' P_1$$

$$X_{12}: \quad {}_1X_{12} = M_m T_2' z_{12} P_{31} + \boxed{(506)P_1}$$

$${}_0X_{12} = M_m T_2' z_{12}' P_{31} + G_m(506)' P_1$$

$$X_{11}: \quad {}_1X_{11} = M_m T_2' z_{11} P_{31} + G_m[(502)F_1 + (504)F_3]P_1 + \boxed{[(501)F_0 + (503)F_2 + (505)]P_1}$$

$${}_0X_{11} = M_m T_2' z_{11}' P_{31} + G_m[(501)F_0' + (503)F_2' + 507]P_1 + \boxed{[(502)F_1' + (504)F_3' + (506) + (507)]P_1}$$

The term $M_m T_2'$ uniquely defines the read-out interval for the core memory, and is true only for Type II orders. The terms enclosed in the dotted brackets are implicit in the configurations of the codes as obtained from the core memory, but are included to indicate the desired logical conditions which apply.

It will be recalled that the N - 1 code bit can also set X_{11} to the one-state; this is not shown here, but is discussed in the section describing the operation of the N Counter Code.

The equations for the end stages of the Y_7 - Y_1 flip-flops and the N_5 - N_1 Counter are noted below.

Y_7: $\quad {}_1y_7 = M_m T_1 X_{11} x_{10} P_{28} + M_m X_{11}' Y_7' Y_6 Y_5 Y_4 Y_3 Y_2 Y_1 P_{28}$

$\quad {}_0y_7 = M_m T_1 X_{11} x_{10}' P_{28} + M_m X_{11}' Y_7 Y_6 Y_5 Y_4 Y_3 Y_2 Y_1 P_{28}$

Y_1: $\quad {}_1y_1 = M_m T_1 X_{11} x_4 P_{28} + M_m X_{11}' Y_1' P_{28}$

$\quad {}_0y_1 = M_m T_1 X_{11} x_4' P_{28} + M_m X_{11}' Y_1 P_{28}$

N_5: $\quad {}_1n_5 = G_m X_{12} x_8 P_{28}$

$\quad {}_0n_5 = G_m X_{12} x_8' P_{28}$

N_1: $\quad {}_1n_1 = G_m X_{12} x_4 P_{28}$

$\quad {}_0n_1 = G_m X_{12} x_4' P_{28}$

The term $M_m X_{11}'$ allows the Y flip-flops to increase their count during Type II orders. Another logic term, $M_m T_1'$, which is not shown here, allows the Y flip-flops to count during Type III orders. The term $M_m T_1$ which controls the transfer of the Y_n Address into the Y flip-flops, prevents this transfer during a Type III order.

4.14 Description of the Major Internal States:

Flip-flops V_2 and V_1 define the four internal states of the computer; in essence, they define the general operational condition of the

machine. The states are listed below:

a. State I: $V_2'V_1'$ Idle

b. State II: $V_2'V_1$ Compute

c. State III: V_2V_1' Fill

d. State IV: V_2V_1 Compute One-Cycle

The names given to each of these internal state conditions describe their functions. Note that State IV, the Compute One-Cycle State, causes the computer to execute one order, as specified by the Order Counter contents. However, because of the command structure of the machine, it is possible for this single order to be very elaborate, should the order refer to a micro-routine.

Possible transitions among the internal states are shown in Figure 9. Tape Code refers to the particular combination of holes in the punched tape which commands the desired State; manual buttons can also be used so that the operator can command the transitions directly from a keyboard.

State I: Idle:

During State I, the Idle State, the computer is prevented from performing computational operations, since the Phase I order search is inhibited (W_c, the search flip-flop, cannot be one-set), and the machine remains locked in the Phase I condition indefinitely.

When first turned on, the computer is constrained to be in the Idle State. This can be accomplished by placing delay relays in the V_2 and V_1 flip-flop circuits so that the flip-flops are zero-set as power is applied to the computer, or by building an unbalance into the circuits so that V_2 and V_1 naturally assume the zero-state when power is applied.

W_c: $\quad {}_1W_c = V_1U_2'U_1'P_{31}$

V_1: $\quad {}_0V_1 = \text{Initial 0-set}$

V_2: $\quad {}_0V_2 = \text{Initial 0-set}$

State II: Compute:

The Compute State, $V_2'V_1$, is set up by the presence of the Compute code on the punched tape. Thereafter, the machine will sequence through its program, remaining in State II until either the Input Command (009) or the Halt Command (012) appears or until a Compute One Cycle Code is read from the tape. The computer then reverts to the Idle State.

$$V_2: \quad {}_0V_2 = (\text{Compute Code})I_1(TR_b)(TP_b)P_{31} + CT_1'(009 + 012)P_{31}$$

$$V_1: \quad {}_1V_1 = (\text{Compute Code})I_1(TR_b)(TP_b)P_{31}$$
$$\quad\quad {}_0V_1 = CT_1'(009 + 012)P_{31}$$

The Compute Code causes V_2 and V_1 to be set to their State II values just before flip-flop I becomes true. This is done, because should the computer be in the Fill State at the time the Compute code is read from the punched-tape, the code itself would be treated as an input. V_2 and V_1 must be altered prior to the one-state of flip-flop I. (See comments in the following section, describing State III.)

State III: Fill:

The Fill State configuration of the V_2 and V_1 flip-flops is set up by a code from the input tape reader. During State III, the computer is

capable of receiving information from the tape reader for processing and storage on the memory. Input information is processed by utilizing micro-routines, and enables the input to be rearranged or modified as required. The computer remains in the Fill State until a Compute or Compute One-Cycle code appears on the input tape, or until an internal Halt command (012) appears during the processing of input information.

In making the transition to the Fill State:

a. The Fill Code on the tape sets up state V_2V_1' at IP_{31}. (Henceforth, V_1' prevents normal computation, just as during the Idle State.)

b. Simultaneously, the Row Select Register, Y_7 through Y_1, is set to the zero-state. Only the end stages are shown below.

c. The Order Type flip-flops, T_2 and T_1, are set to the Type II order configuration, $T_2'T_1$.

$$V_2: \quad {}_1V_2 = (\underline{\text{Fill}}\ \text{code})IP_{31}$$

$$V_1: \quad {}_0V_1 = (\underline{\text{Fill}}\ \text{code})IP_{31}$$

$$Y_7: \quad {}_0Y_7 = (\underline{\text{Fill}}\ \text{code})IP_{31}$$

$$Y_1: \quad {}_0Y_1 = (\underline{\text{Fill}}\ \text{code})IP_{31}$$

$$T_2: \quad {}_0t_2 = (\underline{\text{Fill}}\ \text{code})IP_{31}$$

$$T_1: \quad {}_1t_1 = (\underline{\text{Fill}}\ \text{code})IP_{31}$$

Note that V_2 and V_1 are set to the Fill State configuration at the end of the word interval during which flip-flop I is true, as opposed to the case for the Compute and Compute One-Cycle States, when V_2 and V_1

are set just before flip-flop I becomes true. The reason is as mentioned before. During the Fill State, flip-flop I causes the contents of the H Register to be shifted into the A Register. Therefore, the transition to the Fill State must be made at the end of the I flip-flop interval to prevent the Fill code itself from being fed into the A Register; on the other hand, the transition from the Fill State must be made before entry into the interval when the I flip-flop is true, or else the code denoting the desired non-fill case would be shifted into A.

Once the machine is in the Fill State, each subsequent entry from the tape reader causes the following sequence.

a. The contents of flip-flops H_4 through H_1 of the Input-Output Register are shifted into the A Register, such that the A Register undergoes a shift left of four positions, with the four bits in H_4 through H_1 occupying the four lsd positions of the A Register. This is accomplished by inserting H_4 through H_1 into the recirculation loop of the A Register during the word interval when flip-flop I is true. The logic for only the end stages is shown below.

b. Flip-flop M_m is set to the one-state at IP_{30}, allowing a read-out from the core memory at the address specified by the Row Select Register (which is initially at zero upon entry into the Fill State).

c. The computer is set to the Phase IV condition, U_2U_1, at IP_{31}, so that the first read-out micro-order can be executed.

H_4: $\quad {}_1h_4 = V_2V_1'IA_0P_{31}'$

$\quad\quad {}_0h_4 = V_2V_1'IA_0'$

H_1: $\quad {}_1h_1 = V_2V_1'IH_2P_{31}'$

$\quad\quad {}_0h_1 = V_2V_1'IH_2'$

A_{31}: $1^{a}_{31} = V_2V_1'IH_1$

$0^{a}_{31} = V_2V_1'IH_1'$

M_m: $1^{m}_{m} = V_2V_1'IP_{30}$

U_2: $1^{u}_{2} = V_2V_1'IP_{31}$

U_1: $1^{u}_{1} = V_2V_1'IP_{31}$

As a result, the computer enters the micro-mode, after each input, so that an appropriate micro-routine can be used to process each succeeding input as it is placed into the A Register. (Comments on a possible form of these micro-routines will be found in Part V.)

At the end of the desired micro-routine, the computer returns to Phase I, but can then proceed no further, since V_1' prevails. For each succeeding input, the computer enters the micro-mode, proceeding from the row address contained in the Row Select Register. Therefore, except for the transition into the Fill State when the Row Select Register is zero-set, the micro-routines themselves provide the required starting address in the core for processing each input combination.

State IV: Compute One-Cycle:

The Compute One-Cycle State causes the computer to execute the order specified by the Order Counter and then to revert to the Idle State. The Compute One-Cycle code on the tape is used either to command single cycles during observation, or to allow the operator to halt computation.

The Compute One-Cycle code on the tape does not directly set up State IV, but rather, first establishes the Compute State, $V_2'V_1$. However, G_1 is simultaneously set to the one-state by the One-Cycle code

(the Compute code zero-sets G_1).

V_2: $\quad {}_0V_2 = (\underline{\text{Compute One-Cycle}}\ \text{Code})I_1(TR_b)(TP_b)P_{31}$

V_1: $\quad {}_1V_1 = (\underline{\text{Compute One-Cycle}}\ \text{Code})I_1(TR_b)(TP_b)P_{31}$

G_1: $\quad {}_1G_1 = (\underline{\text{Compute One-Cycle}}\ \text{Code})IP_{31}$

$\quad {}_0G_1 = (\underline{\text{Compute}}\ \text{Code})IP_{31}$

The computer then proceeds with its memory order search during Phase I. When the search is successful, and the machine makes its transition to Phase II, the State flip-flops are set from $V_2'V_1$ to the V_2V_1 condition, under command of flip-flop G_1. Computation during Phases II, III, and IV then proceeds normally. However, as soon as the computer returns to Phase I, the term $U_2'U_1'V_2V_1$ causes the Idle State, $V_2'V_1'$, to become true, halting computation.

V_2: $\quad {}_1V_2 = G_1U_2P_{31}$

$\quad {}_0V_2 = V_2V_1U_2'U_1'P_0$

V_1: $\quad {}_0V_1 = V_2V_1U_2'U_1'P_0$

When a Compute One-Cycle code is used to halt a routine in progress during the Compute State, the receipt of the code causes V_2 to be one-set at U_2P_{31}; the machine will halt when Phase I next occurs.

V MICRO-ROUTINES

A few sample micro-routines are included to illustrate possible operation of the computer. Many of the choices which define the form of the internal information are arbitrary and the programmer is free to modify these at will.

The micro-routines shown here contain the sequences pertinent to the operation being described; for actual use of the machine, these would all be tied together, and, in many instances, a particular sequence of micro-orders would be shared by many micro-routines, with control flip-flops used to control entry and exit points.

Obvious portions of the micro-routines, such as those which deal with pick-up of information from the drum, are omitted.

As mentioned previously, the core address corresponding to the start of a micro-routine can be considered as a pseudo "command" code, so that the headings which describe the micro-routines are also descriptions of available machine "commands".

5.1 Number Representation:

In the micro-routines which follow, positive numbers are represented by magnitude, with an attached zero-bit sign. Negative numbers are represented as binary complements.

5.2 Addition and Subtraction:

To perform signed addition and subtraction, the micro-commands a ADA and s ADA (or, a ADR and s ADR, if preferred) are used directly. To add

and subtract absolute values, negative numbers are first decomplemented.

5.3 Multiplication:

Product Sign Determination:

To perform multiplication, the product sign is first determined. The actual multiplication process uses magnitudes for the numbers, independent of sign. At the completion of the multiplication process, the product is complemented if the original sign determination indicates a negative product.

The micro-routine for determining the sign of the product is shown in Table 5. The multiplier is originally in the A Register, the multiplicand in D. At the completion of this micro-routine, F_0 will be in the zero-state for a positive product or in the one-state for a negative result. Both A and D will contain magnitudes, with zero-bits in the sign position.

The time required is 3 word-times for plus-plus, 6 word-times for plus-minus, 3 word-times for minus-plus, and 6 word-times for minus-minus.

Multiplication Process:

The actual multiplication micro-routine is shown in Table 6. To multiply two 31 digit numbers requires 64 word-times. The routine as shown gives a double length positive product, or a non-rounded-off single length negative product. (If the convention of magnitude and sign is adopted, the proper sign could be attached directly to the product, eliminating the comp A operation. Double-length products of either sign would then be obtained.)

To obtain a full double-length complemented negative product in the

A and R Registers, the micro-routine shown in Table 7 should be superimposed upon that of Table 6. The micro-routine inspects the msd of R before and after the contents of R are complemented. If the msd of R remains zero when R is complemented, the direct complement of A is correct. If the original msd of R is one, or if it changes from zero to one when R is complemented, a one-bit must be subtracted from the lsd position of the complemented A Register. The additional time required to process a negative double-length product is either four or seven word-times.

To obtain a single-length product with round-off, whether positive or negative, the micro-routine shown in Table 8 should be superimposed upon that indicated in Table 6. A round-off product requires a total of either 68 or 69 word-times.

5.4 Division:

Quotient Sign Determination:

The sign of the quotient is determined as in multiplication, and uses the identical micro-routine (Table 5).

Division Process:

The micro-routine shown in Table 9 causes the double-length number in the A and R Registers to be divided by the contents of D. As before, all numbers are expressed as magnitudes, the sign of the quotient having first been determined by using the micro-routine of Table 5. The resulting quotient is placed in the A Register, complemented if negative, with the remainder in R.

At the start of the micro-routine, a test is made to insure that the magnitude of the dividend is less than that of the divisor, since the

quotient is constrained to be less than one in magnitude. If the divisor is less than the dividend, the micro-routine immediately exits from micro-order 3; at the option of the programmer, the machine may be made to halt, suitable shifts can be made, etc.

The process used is the familiar von Neumann non-restoring method. The requirements are summarized below.

a. If subtracting D from A and R:

   No Carry: quotient bit is one; next operation is subtract.

   Carry: quotient bit is zero; next operation is add.

b. If adding D to A and R:

   No Carry: quotient bit is zero; next operation is add.

   Carry: quotient bit is one; next operation is subtract.

After each add or subtract operation, the contents of A and R are left-shifted, and the quotient bit is inserted into the lsd position of R.

Some comments on the micro-routine are noted below. The numbers refer to the micro-orders.

1-3: Checks to insure that $|D| > |A|$.

5,6: Add cycle.

8,9: Subtract cycle.

11-14: Final quotient bit; also, compensates for an extra add-subtract operation.

15-17: Transfers quotient to A, remainder to R. If quotient is required to be negative, A is complemented.

The division process requires from 72 to 104 word-times, depending upon the configuration of the quotient (consecutive zeroes or ones in the quotient require an extra word-time to process).

5.5 Conditional Commands:

Conditional commands generally cause a computer to choose one of two paths, as a function of some relation of numbers in the machine. For a single-address computer, the choice is usually between executing the next order as specified by the Order Counter, or executing the order to be found at a given address.

For this computer, the choice is represented by either of two final conditions; the micro-routine exits at a rmc (201) micro-command, or exits after executing micro-command 215, tr MrOC.

An extremely large variety of conditional commands can be synthesized. A few are shown below.

Sign of the A Register:

Table 10 illustrates how the machine will proceed to the order at location m if the A Register contents are negative; Table 11 shows the converse, i.e., branch if A is positive or zero. These can easily be modified to inspect either of the other registers by direct inspection of the sign position or by using the exchange micro-commands. The conditional commands shown require either two or three word-times.

Magnitude Comparison:

To branch as a function of the relative absolute magnitudes of two quantities, it is necessary to perform a simplified version of the micro-routine shown in Table 5 to express both numbers as magnitudes. A subtraction is then performed, and the sign of the difference is used to determine the branching condition.

Zero Inspection:

The zero-state of a number can be determined by placing the number

into the D Register, and then subtracting it from the cleared A Register. By using K to set a Control Flip-Flop, a branching in either direction can be commanded, i.e., a command, "Change on Non-Zero" or a command, "Change on Zero" can be synthesized.

5.6 Binary Square Root:

The square root of a binary number can be extracted by using the micro-routine shown in Table 12. This is not an iterative process, but rather, inspects the number bit-by-bit, determining successive digits of the square root in the process.

The procedure is derived in the Appendix. The comments below indicate the mechanical process followed in performing the micro-routine.

The number whose root is to be extracted is assumed to be in the A Register, expressed as a magnitude. As digits of the root are derived, they are inserted into the lsd end of the A Register. The net result is a 15-digit root of a 30-bit number. (To extend the process to double-length numbers, it is necessary to store the first result, and then obtain from memory, the remaining half of the double-length number; the process is straightforward, and is not shown here.) Using the following nomenclature (see Appendix):

r_i = position of the trial bit of the root

C_i = the correction term applied to the remainder,

the mechanical process is as follows.

a. If <u>subtracting</u> the correction term:

<u>Carry</u>: root-bit is zero; the next operation is <u>add</u>;

the correction term is $C_{i+1} = \frac{1}{2}[C_i + \frac{1}{2}(\Delta r_i)^2]$

No Carry: root-bit is one; the next operation is subtract;

the correction term is $C_{i+1} = \frac{1}{2}[C_i + \frac{3}{2}(\Delta r_i)^2]$

b. If adding the correction term:

Carry: root-bit is one; the next operation is subtract;

the correction term is $C_{i+1} = \frac{1}{2}[C_i - \frac{1}{2}(\Delta r_i)^2]$

No Carry: the root-bit is zero; the next operation is add;

the correction term is $C_{i+1} = \frac{1}{2}[C_i - \frac{3}{2}(\Delta r_i)^2]$

The A Register contains the residue which results from the successive trial additions and subtractions. The D Register contains the C_i term; R contains a generating-bit which is used to modify the C_i value for successive trials. The $\frac{1}{2}$ factor common to all of the C_i terms is provided by the relative shifts of the A and D Registers.

The operation of the micro-routine is briefly described below, with comments referring to the micro-orders named:

1-5: A contains the required number. R and D contain 0.01.

6,7: The subtraction sequence.

8-10: Preparation for generating C_i after each operation.

11-15: Makes the correction for the post-add case.

16,17: The addition sequence.

18-23: Makes the correction for the post-subtract case.

24-27: Final root-bit is inserted; prepares for positioning of the root.

28-29: Shifts A so that the root occupies the 15 digit positions after the sign position.

The square-root micro-routine shown requires between 158 and 188 word-times, or less than three drum-revolutions.

5.7 Input:

During the Fill State, the computer enters the Micro-Mode after each input from the tape reader. The micro-program facility is used to process input information, and allows the insertion of any of the order types, numerical information, etc. in format convenient for the user; using micro-routines, the computer then converts the input into the proper form for internal use. The comments below describe methods which are applicable for processing the various inputs; although most of the inputs share similar micro-sequences, the interleaving of these routines to economize on core storage will not be shown here.

Fill Address Determination:

The Fill code, which sets up the Fill State, also causes the Row Address Register to be zero-set. The first set of inputs is then used to define the address location into which information is to be filled, with the B Register used thereafter to advance the fill position. The micro-routine which determines the fill address position must therefore begin at position 0 in the core memory.

At any subsequent time in the input process, if the memory position is required to change other than monotonically, the Fill code will cause the machine to return to position 0 of the core, causing the subsequent input to be interpreted as a fill address.

The fill address is specified by four decimal digits interpreted as a channel location and a sector location. This address (converted to

12 binary bits) is written into the Address Register. The micro-routine shown in Table 13 accomplishes this. The process required to insert a one-bit into the B Register is not shown here; it is very straightforward.

The comments below apply to the micro-orders mentioned.

0-2: Preparatory operations. The machine idles after micro-order 2, awaiting the input from the tape reader.

3-9: Decimal to binary conversion procedure. At the end of this process, the binary number is in the D Register. In detail:

3: Preliminary sum is in A.

4,5: Preliminary sum is in both A and D. (A micro-command, tr AD, would be very convenient here.)

6-8: Causes A to contain the preliminary sum, multiplied by binary 1010 (decimal ten).

9: Puts result into A; also checks tally of N Counter.

10-13: Causes shift left of 6 positions of the binary equivalent of the first two decimal digits (hence, the channel address).

14-17: Causes the entire address to be left shifted seven positions (hence, to the required address position), and then to be placed into the Address Register.

The computer exits to the micro-order position corresponding to the start of the micro-routine next described, which interprets subsequent inputs as orders, numerical information, etc., and processes accordingly.

The maximum length sequence between inputs from the tape is 19 word-times, or less than 6 milliseconds. Therefore, no conflict with the tape reader cycle will occur.

Interpreting Input Words:

After the initial fill address has been determined, the computer

receives input information and processes it before storing it on the memory. To differentiate among the various kinds of input words, each input word would be preceded by a symbol with mnemonic significance. Numerical information would be preceded by an arithmetic sign, orders could be preceded by an appropriate letter, etc.

For illustrative purposes, the following codes for the symbols will be assumed:

001 $\begin{cases} 1 \\ 0 \end{cases}$ denotes numerical information. (Final bit represents positive or negative.)

010 $\begin{cases} 1 \\ 0 \end{cases}$ denotes Order input. (Final bit denotes the B bit.)

100 $\begin{cases} 1 \\ 0 \end{cases}$ denotes a Micro-Order to be stored on the drum. (Final bit represents the End-of-Transfer bit.)

The computer receives one of the above code symbols and branches so that subsequent inputs are converted or arranged in the proper manner. The micro-routine required to interpret the above sample codes is shown in Table 14, and is discussed below.

After the input, the four bits in the lsd position of the A Register represent the word interpretation code. The lsd of the code is stored in F_0, while the remaining bits are successively transferred into F_1 and control the exit point of the micro-routine. The α addresses represent the starting positions for micro-routines which process the following classes of input:

α_1: Numerical information.

α_2: Orders.

α_3: Micro-orders to be stored on the drum.

α_4: Optional.

The interpretation requires from two to five word-times, depending upon the exit point. The micro-routines required at the various α address locations are described below.

Numerical Information:

The micro-routine which begins in location α_1 is shown in Table 15. Decimal inputs are converted to binary. Simultaneously, the binary equivalent of 10^n is generated, where n is the number of decimal digit inputs. At the end of the input word, a division is performed to convert the number to magnitude less than one. If the number is negative, it is then complemented.

The process can either require fixed length inputs (that is, a fixed number of decimal digits is always provided for each word), or a "finish" code can be used to denote the end of significant digit input. The former approach would use the N Counter to count the number of inputs after the sign and would initiate the division process automatically; the latter approach would require the "finish" code to be a configuration of four binary digits not used to represent a decimal number and would inspect each input prior to processing to determine the presence of a "finish" code. Both methods are straightforward; the latter approach is shown in Table 15.

At the beginning of the micro-routine, R and D are assumed to be in the zero-state as a result of the input interpretation micro-routine of Table 14. Assume that the "finish" code, which represents the end of a numerical input word, is represented by the code 1010 (does not correspond to any decimal number). The micro-order groups are described below.

1-2: Preparatory actions. Computer idles after 2, awaiting the input.

3-4: Part of the inspection of the input to look for the finish code.

5: Puts input back into original position in A, if the second lsd is 0 (i.e., the input was not a finish code).

6-11: Decimal-Binary conversion procedure:

6-8: Puts number into both A and D.

9-11: Multiplies by 1010.

11-12: If F_1 is zero, have generated only the number itself. Therefore, return to 7 to generate 10^n. If F_1 is one, have obtained both quantities.

13-14: Return to 2 to await the next input. Number is in D, 10^n is in R.

15-17: Test for finish code.

18-19: If finish code, put the number in A, 10^n into D, clear R. Exit to the Division micro-routine; initial conditions are as required by Table 9.

After the division process, the A Register contains the binary equivalent of the decimal input, with magnitude less than one, and complemented if negative. This value is stored into the drum memory, and the Row Address Register is set to the value corresponding to the starting point of the micro-routine in Table 14, preparing, therefore, for the next input word.

The maximum-length micro-routine between inputs is 24 word-times. At the end of the word, the maximum time required to determine the presence of the finish code is 6 word-times, so that, even with a full 104 word-time division, the total time prior to storage on the drum would be less than two drum revolutions. Even assuming a full drum revolution to find the desired memory position, the entire process requires less than three drum revolutions.

Order Input:

If the machine receives a code denoting an order input (exit at α_2 in Table 14), the computer inspects the following input to determine which of the Order Types is involved. Types I, II, and III are processed identically (if the command code in Type I is expressed by three decimal digits; 002, for example). Type IV has a slight variation.

The micro-routine required to determine the Order Type is shown in Table 16. Micro-orders 1-4 are executed immediately following the micro-routine of Table 14, i.e., α_2 is micro-order 1 of Table 16. The machine idles, receives the order-type code, and proceeds from micro-order 5. Exit at 7 to β_1 denotes Type I, II, or III; exit at 9 to β_2 denotes Type IV.

Type I, II, or III Input:

The micro-routine for a Type I, II, or III order input is shown in Table 17. Initially F_0 is in the zero-state because of the effect of the micro-routine in Table 16.

1: Sets F_1 to zero, N to 2.

2: Clears D; machine idles, awaiting input.

3-9: Decimal-binary conversion, as before. Under control of N, the machine returns to 2 each time, after converting each input.

10: Inspects F_0. Proceeds to 11 or 18.

11: Sets N to 7 to control a later left-shift of 8 places.

12-14: Left-shifts A under control of N, adds D, and puts the sum into R. This operation incorporates the latest input into the order word. Also, inspects F_0 and proceeds to 15 or 16.

15: Used after input of command code or Y_n Address. One-sets F_0.

16: Inspects F_1. Proceeds to 17 or 20.

17: Used after channel address input. Sets F_1 to one.

18-19: Sets N to 5 for later shift-left of 6 places. Used after both the channel and sector address inputs.

20-22: Final shift of 7 places after the sector address input. The micro-routine exits to store the order in the required memory position.

After the order is stored into memory, the Row Address Register is set to the start of the micro-routine in Table 14. The maximum-length micro-routine between inputs requires 20 word-times. At the end of the order input, a maximum of 28 word-times are required; allowing a drum revolution for memory access still allows ample time between tape inputs.

Type IV Input: The micro-routine required to process a Type IV order will not be shown in detail here. The sequence is very similar to Table 17, except that the first two pairs of decimal digits are interpreted as the Arithmetic and Control Micro-Commands, and are converted to binary and placed in the order word as shown in Figure 2.

Micro-Order Input:

If the machine receives a code denoting input of a micro-order to be stored on the magnetic drum (exit at α_3 in Table 14), the computer is required to process the input word shown in Figure 3. The input from the tape will consist of the following:

a. An initial code, denoting that the word is a micro-order (the final one or zero bit is the End-of-Transfer bit).

b. A decimal one or zero, to denote the N Code bit.

c. Two decimal digits, denoting the Arithmetic Micro-Command.

d. Two decimal digits, denoting the Control Micro-Command.

e. Three successive decimal digits, representing the Control Input,

Control Selector, and Y_n Control Codes respectively.

f. Three decimal digits, representing the Y_n Address.

Inputs c, d, and f require a decimal-binary conversion.

The detailed micro-routine will not be shown here, but a possible method is described. Input a, which is actually processed by Table 14, supplies a one or a zero bit in the lsd position of A. Input b requires no processing, since it automatically assumes the proper position.

Inputs c and d are converted from decimal to binary; the N Counter is used to separate the decimal input into pairs, and to cause two traversals of the decimal-binary sequence. F_0 can be used to separate between the two pairs of inputs. After each two-decimal input, the order word is shifted left four positions, and the converted-to-binary micro-command is added.

After input d, let both F_0 and F_1 be in the one-state, and then sequence through the combinations F_1F_0, $F_1{}'F_0$, and $F_1{}'F_0{}'$. The first input (of group e) causes the order word to be left-shifted by three places before the input is added. Prior to the addition of the next input, the one-state of F_1 causes a two-position shift; simultaneously, F_1 is zero-set. After the third input of group e, the zero-state of F_1 causes a left-shift of three before the input is added. F_0 is zero-set at the start of the processing of the third input, and informs the machine that the inputs to follow are the Y_n Address digits.

The three decimal digits which represent the Y_n Address are converted and attached to the order word in the manner shown in Table 17.

Since the main time-consuming process here is decimal-binary conversion, the required time per input is much less than the interval available.

VI THE LOGICAL EQUATIONS

6.1 Comments on the Form of the Equations:

The logical equations for the computer comprise Part 6.2 of this thesis. The general comments below concern the form of the equations.

First of all, the equations are presented in unmechanized form, i.e., they have not, as yet, been modified to conform to the restrictions imposed by particular gating circuits. The choice of circuitry directly affects the mechanization procedure. Without first determining the allowable load per flip-flop, the permissible cascading of logic terms, the form of the gating (and-or gating, etc.), there is little benefit to be gained in establishing arbitrary criteria, and then mechanizing accordingly.

Secondly, the terms in the equations have not been extensively manipulated or simplified. These operations are properly left to be performed with the mechanization. Also, these processes make the final equations more difficult to understand; at present, the logic is presented in more or less natural form, and better illustrates the functional nature of each of the terms.

Some possible simplifications are mentioned below, however, to illustrate that in the final mechanization, many of the terms will appear less formidable in length, though perhaps more obscure in meaning.

Code Simplifications: The codes representing the various commands and micro-commands were initially selected with care to simplify the logic required to express the combinations. A few examples are shown here.

a. Long arrays of code numbers are often very simple.

Example: the following appears as part of ${}_0u_2$:

$$G(002 + 003 + 006 + 007 + 008 + 010 + 011 + 014 + 015) = G[S_2 + S_4S_3'S_1']$$

b. Many codes appear in pairs. Where possible, these are selected as consecutive even-odd numbers, or share bits in common.

Example: in ${}_1r_{31}$:

$$G_m(104 + 105) = G_mQ_4'Q_3Q_2'$$

At the time of mechanization, it would be advisable to re-examine the code assignment to note whether simplifications may be possible under the conditions imposed by the mechanization conventions.

Redundancies: Redundant terms often help interpret and sometimes simplify the logic for particular mechanizing conditions. Hence, no systematic efforts have been made to eliminate all of them.

Core Memory Output Register: In obtaining information from the core memory, the 24 bits of a row must be temporarily stored. Since many of the core output bits are needed only once, however, it should be possible to store these in memory-type core elements, rather than in relatively expensive flip-flops, and then to sense these auxiliary core elements at the required time. Columns in the core for which this technique appears applicable are: 27, 18-16, 13-11, and 10-4. In the logical equations, core memory outputs are labeled as z_n, where n represents the column. The symbol x_n is used to denote possible auxiliary core elements; if this technique is not used, then X_n can be substituted, where X_n represents a flip-flop storing the column output.

6.2 The Logical Equations:

Phase Control:

U_1: $$_1u_1 = U_1'W_cP_{31} + T_2'T_1M_mU_1'P_{31} + GU_1'(006' + 013 + 014 + 015)P_{31} + V_2V_1'IP_{31}$$

$$_0u_1 = (T_2' + T_1')U_2'U_1P_{31} + G_m(X_{13} + 201 + T_2)P_{31} + GU_1[002 + 003 + 006 + 007 + 008 + 010 + 011 + 014 + 015 + M_4'(004 + 005) + (O_c + W_c)(013)]P_{31} + T_2T_1'U_1M_mWP_{31}$$

U_2: $$_1u_2 = U_2'U_1P_{31} + V_2V_1'IP_{31}$$

$$_0u_2 = GT_1'U_1[002 + 003 + 006 + 007 + 008 + 010 + 011 + 014 + 015 + M_4'(004 + 005) + (O_c + W_c)(013)]P_{31} + GT_1'(009 + 012)P_{31} + G_m(201 + T_2)P_{31} + T_2T_1'U_1M_mWP_{31}$$

Word Control:

W_c: $$_1w_c = V_1U_2'U_1'P_{31} + T_2T_1'U_2U_1'O_c'P_{31} + M_mT_1X_{13}'P_{31} + GU_1(013)W_c'O_c'P_{31} + GU_1'O_c'(002 + 003 + 004 + 005 + 007 + 008 + 010 + 011)P_{31}$$

$$_0w_c = U_2'U_1'(O_0S_c' + O_0'S_c)(Ch_0P_{9-7} + Ch_0'P_{12-7}) + U_2U_1'(M_0S_c' + M_0'S_c)\cdot(Ch_0P_{9-7} + Ch_0'P_{12-7}) + G_m(101)A_0P_{31}' + G_m(X_{13} + 201 + T_2)P_{31} + (T_2' + T_1')U_2'U_1P_{31} + GU_1W_cP_{31} + T_2T_1'U_1M_mWP_{31}$$

Order Storage Control:

$$^{o}c: \quad {}_{1}{}^{o}c = U_2'U_1'W_cP_{31} + U_2'U_1T_1'M_0P_{31} + GU_1'(010)W_cP_{31} + GU_1'(013 + 015)P_{31} + G_m(201)P_{31}$$
$$+ G_mT_2'X_{13}z_{23}P_{31} + T_2T_1'U_1M_mWP_{31}$$

$${}_{0}{}^{o}c = U_2'U_1{}^{o}c{}^{o}{}_{0}'P_{18-7} + GU_1'(013)'(015)'{}^{o}cP_{31} + GU_1[010\ {}^{o}{}_{0}' + 013\ B_0 + 015\ B_0']{}^{o}cP_{18-7}$$
$$+ U_2U_1'T_2{}^{o}cP_{31} + U_2'U_1(M_0' + T_1)P_{31}$$

Order Type Selector:

$$T_1: \quad {}_{1}t_1 = U_2'U_1WP_{27-0} + (\underline{\text{Fill}}\ \text{Code})IP_{31}$$

$${}_{0}t_1 = U_2'U_1W'P_{27-0}$$

$$T_2: \quad {}_{1}t_2 = U_2'U_1WE_2'E_1'$$

$${}_{0}t_2 = U_2'U_1W'E_2'E_1' + (\underline{\text{Fill}}\ \text{Code})IP_{31}$$

Micro-Mode Indicator:

$$M_m: \quad {}_{1}m_m = T_2'T_1U_2U_1'X_{13}'P_{30} + T_2T_1'U_2U_1{}^{o}c'P_{31} + T_2T_1U_2'U_1P_{31} + GT_1U_1[002 + 003 + 006 + 007$$
$$+ 008 + 010 + 011 + 014 + 015 + N_4'(004 + 005) + ({}^{o}c + W_c)(013)]P_{31} + V_2V_1'IP_{30}$$

$${}_{0}m_m = G_m(X_{13} + 201 + T_2)P_{31} + M_mT_1'{}^{o}cP_1 + M_mT_2'{}^{o}cP_{16}$$

A Register Write:

$$A_{31}: \quad {}_1a_{31} = G_m(101)(W_cA_0 + W_c'A_0') + G_m(102 + 103)(203 + 205 + 207)'A_s + G_m(115)P_{31}$$
$$+ G_m(102 + 103)'(106)'(202 + 204 + 206)A_1P_{31}' + G_m(106)'(114)'(204)A_0P_{31}$$
$$+ G_m(114\ P_{31})'(106)'[(203 + 205 + 207)C_1P_0' + (210 + 211)R_0 + (212)D_0]$$
$$+ G_m(106)'(207)R_{31}P_0 + GU_1[(008)WA_0 + (014)B_0] + V_2V_1'IH_1$$
$$+ [G' + U_1' + (008)'(014)'][V_2' + V_1' + I'][G_m' + (101 + 102 + 106 + 114\ P_{31})'\cdot$$
$$\cdot(202 + 203 + 204 + 205 + 206 + 207 + 210 + 211 + 212)']A_0$$

$${}_0a_{31} = G_m(101)(W_cA_0' + W_c'A_0) + G_m(102 + 103)(203 + 205 + 207)'A_s' + G_m(106)P_0$$
$$+ G_m(102 + 103)'(202 + 204 + 206)A_1'P_{31}' + G_m(115)'(202 + 206)P_{31}$$
$$+ G_m(115\ P_{31})'[(203 + 205 + 207)C_1'P_0' + (210 + 211)R_0' + (212)D_0']$$
$$+ G_m(114)P_{31} + G_m(203)P_0 + G_m(207)R_{31}'P_0 + G_m(115)'(204)A_0'P_{31}$$
$$+ GU_1[(008)(W' + A_0') + (014)B_0'] + V_2V_1'IH_1' + [G' + U_1' + (008)'(014)']\cdot$$
$$\cdot[V_2' + V_1 + I'][(101 + 102 + 106 + 115\ P_{31})'(202 + 203 + 204 + 205 + 206$$
$$+ 207 + 210 + 211 + 212)' + G_m']A_0'$$

A Register Read:

A_0: $\quad {}_1a_0 = G_m(204 + 206)A_1P_{31} + [G_m' + (204)'(206)']A_1$

$\quad {}_0a_0 = G_m(204 + 206)A_1'P_{31} + [G_m' + (204)'(206)']A_1'$

R Register:

R_{31}: $\quad {}_1r_{31} = G_m(104 + 105)A_s + G_m(113)P_{31} + G_m(107)'(206)[R_1P_{31}' + A_0(112)'P_{31}]$

$+ G_m(107)'(112\ P_{31})'[(207)\ P_0'C_2 + (209 + 211)A_0 + (213)D_0]$

$+ [G_m' + (104 + 105 + 107 + 112\ P_{31})'(206 + 207 + 209 + 211 + 213)']R_0$

$\quad {}_0r_{31} = G_m(104 + 105)A_s' + G_m(107)P_0 + G_m(112)P_{31} + G_m(206)[R_1'P_{31} + (113)'A_0'P_{31}]$

$+ G_m(113\ P_{31})'[(207)(P_0 + C_2') + (209 + 211)A_0' + (213)D_0']$

$+ [G_m' + (104 + 105 + 107 + 113\ P_{31})'(206 + 207 + 209 + 211 + 213)']R_0'$

R_0: $\quad {}_1r_0 = G_m(111)P_{31} + [G_m' + (110)' + P_{31}']R_1$

$\quad {}_0r_0 = G_m(110)P_{31} + [G_m' + (111)' + P_{31}']R_1'$

D Register:

$$D_{31}: \quad {}_1d_{31} = G_m(108)'(109\ P_{31})'[(212)A_0 + (213)R_0] + GU_1(003)W$$
$$+ [G' + (003)'][G_m' + (108 + 109\ P_{31})'(212 + 213)']D_0$$

$${}_0d_{31} = G_m[(108)P_0 + (109)P_{31} + (212)A_0' + (213)R_0'] + GU_1(003)W'$$
$$+ [G' + (003)'][G_m' + (108)'(212 + 213)']D_0'$$

Address Register:

$$M_{31}: \quad {}_1m_{31} = U_2'U_1W + U_2U_1'O_c(B_0'M_0'K + B_0'M_0K' + B_0M_0'K' + B_0M_0K)P_{18-7} + G_m(208)A_0P_{18-7}$$
$$+ [G_m' + (208)'](U_2 + U_1')(U_2' + U_1 + O_c')M_0$$

$${}_0m_{31} = U_2'U_1W' + U_2U_1'O_c(B_0M_0K' + B_0M_0'K + B_0'M_0K + B_0'M_0'K')P_{18-7} + G_m(208)A_0'P_{18-7}$$
$$+ [G_m' + (208)'](U_2 + U_1')(U_2' + U_1 + O_c')M_0'$$

Order Counter:

$$O_{31}: \quad {}_1o_{31} = U_2'U_1(O_cO_0' + O_c'O_0) + GU_1(013)W_cM_0P_{18-7} + G_m(215)M_0P_{18-7}$$
$$+ [G_m' + (215)'](U_2 + U_1')[G' + U_1' + (013)' + W_c']O_0$$

$${}_0o_{31} = U_2'U_1(O_cO_0 + O_c'O_0') + GU_1(013)W_cM_0'P_{18-7} + G_m(215)M_0'P_{18-7}$$
$$+ [G_m' + (215)'](U_2 + U_1')[G' + U_1' + (013)' + W_c']O_0'$$

B Register:

B_{31}: $${}_1b_{31} = GU_1(011)WP_{18-7} + GU_1(013 + 015)({}_0c\,B_0' + {}_0c'\,B_0)P_{18-7}$$
$$+ [G' + U_1' + (011)'(013)'(015)']B_0$$

$${}_0b_{31} = GU_1(011)W'P_{18-7} + GU_1(013 + 015)({}_0c\,B_0 + {}_0c'\,B_0')P_{18-7}$$
$$+ [G' + U_1' + (011)'(013)'(015)']B_0'$$

Memory Write f.f. for Channel n: $(n \neq 0)$:

M_{wn}: $${}_1m_{wn} = f_n(C_6 - C_1)GU_1[(002)A_0 + (005)M_{r0} + (007)A_0P_{18-7} + (010)({}_0c\,0_0' + {}_0c'\,0_0)P_{18-7}]$$
$${}_0m_{wn} = f_n(C_6 - C_1)GU_1[(002)A_0' + (005)M_{r0}' + (007)A_0'P_{18-7} + (010)({}_0c\,0_0 + {}_0c'\,0_0')P_{18-7}]$$

Short Loop Write: $(n = 0)$:

M_{w0}: $${}_1m_{w0} = f_0(C_6 - C_1)GU_1[(002)A_0 + (005)M_{r0} + (007)A_0P_{18-7} + (010)({}_0c\,0_0' + {}_0c'\,0_0)P_{18-7}]$$
$$+ GU_1(004)W$$
$$+ [G' + U_1' + (004)'\{f_0'(C_6 - C_1) + (002)'(005)'[(007)'(010)' + P_{18-7}']\}]M_{r0}$$

$${}_0m_{w0} = f_0(C_6 - C_1)GU_1[(002)A_0' + (005)M_{r0}' + (007)A_0'P_{18-7} + (010)({}_0c\,0_0 + {}_0c'\,0_0')P_{18-7}]$$
$$+ GU_1(004)W'$$
$$+ [G' + U_1' + (004)'\{f_0'(C_6 - C_1) + (002)'(005)'[(007)'(010)' + P_{18-7}']\}]M_{r0}'$$

Channel Select Register:

(A Delay)

C_1: $\quad {}_1c_1 = U_1 \cdot C_2 P_{18-7} + G_m[(102 + 103)A_s + (102 + 103) \cdot A_0]$

$\quad\quad {}_0c_1 = U_1 \cdot C_2 \cdot P_{18-7} + G_m[(102 + 103)A_s \cdot + (102 + 103) \cdot A_0 \cdot]$

(R Delay)

C_2: $\quad {}_1c_2 = U_1 \cdot C_3 P_{18-7} + G_m(207)R_0$

$\quad\quad {}_0c_2 = U_1 \cdot C_3 \cdot P_{18-7} + G_m(207)R_0 \cdot$

C_3: $\quad {}_1c_3 = U_1 \cdot C_4 P_{18-7}$

$\quad\quad {}_0c_3 = U_1 \cdot C_4 \cdot P_{18-7}$

C_4: $\quad {}_1c_4 = U_1 \cdot C_5 P_{18-7}$

$\quad\quad {}_0c_4 = U_1 \cdot C_5 \cdot P_{18-7}$

C_5: $\quad {}_1c_5 = U_1 \cdot C_6 P_{18-7}$

$\quad\quad {}_0c_5 = U_1 \cdot C_6 \cdot P_{18-7}$

(R Delay) (Con't)

C_6: $1^{c_6} = U_2 'U_1 '^{O}_{0} P_{18-7} + U_2 U_1 '^{M}_{0} P_{18-7}$

$0^{c_6} = U_2 'U_1 '^{O}_{0} 'P_{18-7} + U_2 U_1 '^{M}_{0} 'P_{18-7}$

Carry f.f.:

K: $1^{k} = U_2 U_1 '^{B}_{0} M_0 P_{18-7} + C_m(202 + 206) '[(102 + 104)A_0 D_0 + (103 + 105)A_0 'D_0]P_{31} '$

$+ C_m(202 + 206)[(102 + 104)A_1 D_0 + (103 + 105)A_1 'D_0]P_{31} '$

$0^{k} = P_{31} + U_2 U_1 '^{B}_{0} 'M_0 ' + C_m(202 + 206) '[(102 + 104)A_0 'D_0 ' + (103 + 105)A_0 D_0 ']$

$+ C_m(202 + 206)[(102 + 104)A_1 'D_0 ' + (103 + 105)A_1 D_0 ']$

Control Micro-Command Register:

Q_1: $1^{q_1} = U_2 'U_1 Q_2 P_{27-0} + M_m T_2 'z_{23} P_{31}$

$0^{q_1} = U_2 'U_1 Q_2 'P_{27-0} + M_m T_2 'z_{23} 'P_{31}$

Q_2: $1^{q_2} = U_2 'U_1 Q_3 P_{27-0} + M_m T_2 'z_{24} P_{31}$

$0^{q_2} = U_2 'U_1 Q_3 'P_{27-0} + M_m T_2 'z_{24} 'P_{31}$

Control Micro-Command Register: (Con't)

Q_3: $1^{q}_3 = U_2'U_1Q_4P_{27-0} + M_mT_2'z_{25}P_{31}$

$0^{q}_3 = U_2'U_1Q_4'P_{27-0} + M_mT_2'z_{25}'P_{31}$

Q_4: $1^{q}_4 = U_2'U_1T_1P_{27-0} + M_mT_2'z_{26}P_{31}$

$0^{q}_4 = U_2'U_1T_1'P_{27-0} + M_mT_2'z_{26}'P_{31}$

Command and Arithmetic Micro-Command Register:

S_1: $1^{s}_1 = U_2'U_1S_2P_{27-0} + M_mT_2'z_{19}P_{31}$

$0^{s}_1 = U_2'U_1S_2'P_{27-0} + M_mT_2'z_{19}'P_{31}$

S_2: $1^{s}_2 = U_2'U_1S_3P_{27-0} + M_mT_2'z_{20}P_{31}$

$0^{s}_2 = U_2'U_1S_3'P_{27-0} + M_mT_2'z_{20}'P_{31}$

S_3: $1^{s}_3 = U_2'U_1S_4P_{27-0} + M_mT_2'z_{21}P_{31}$

$0^{s}_3 = U_2'U_1S_4'P_{27-0} + M_mT_2'z_{21}'P_{31}$

Command and Arithmetic Micro-Command Register: (Con't)

S_4: $1^{S_4} = U_2'U_1Q_1P_{27-0} + M_mT_2'z_{22}P_{31}$

$0^{S_4} = U_2'U_1Q_1'P_{27-0} + M_mT_2'z_{22}'P_{31}$

Core Read-Out f.f.:

X_{27}: $1^{X_{27}} = M_mT_2'z_{27}P_{31}$

$0^{X_{27}} = M_mT_2'z_{27}'P_{31}$

X_{13}: $1^{X_{13}} = M_mT_2'z_{13}P_{31}$

$0^{X_{13}} = M_mT_2'z_{13}'P_{31} + G_m(507)'P_1 + CU_1$

X_{12}: $1^{X_{12}} = M_mT_2'z_{12}P_{31}$

$0^{X_{12}} = M_mT_2'z_{12}'P_{31} + G_m(506)'P_1$

X_{11}: $1^{X_{11}} = M_mT_2'z_{11}P_{31} + G_m[(502)F_1 + (504)F_3]P_1 + G_mX_{27}(N_5 + N_4 + N_3 + N_2 + N_1)P_1$

$0^{X_{11}} = M_mT_2'z_{11}'P_{31} + G_m[(501)F_0' + (503)F_2' + (507)]P_1 + U_2'P_{31}$

Row Select Register:

Y_1: $\quad {}_1y_1 = U_2'U_1Y_2P_{25-0} + M_mT_1X_{11}x_4P_{28} + M_m(X_{11}' + T_1')Y_1'P_{28}$

$\quad {}_0y_1 = U_2'U_1Y_2'P_{25-0} + M_mT_1X_{11}x_4'P_{28} + M_m(X_{11}' + T_1')Y_1P_{28} + (\underline{\text{Fill}}\ \text{Code})IP_{31}$

Y_2: $\quad {}_1y_2 = U_2'U_1Y_3P_{25-0} + M_mT_1X_{11}x_5P_{28} + M_m(X_{11}' + T_1')Y_2'Y_1P_{28}$

$\quad {}_0y_2 = U_2'U_1Y_3'P_{25-0} + M_mT_1X_{11}x_5'P_{28} + M_m(X_{11}' + T_1')Y_2Y_1P_{28} + (\underline{\text{Fill}}\ \text{Code})IP_{31}$

Y_3: $\quad {}_1y_3 = U_2'U_1Y_4P_{25-0} + M_mT_1X_{11}x_6P_{28} + M_m(X_{11}' + T_1')Y_3'Y_2Y_1P_{28}$

$\quad {}_0y_3 = U_2'U_1Y_4'P_{25-0} + M_mT_1X_{11}x_6'P_{28} + M_m(X_{11}' + T_1')Y_3Y_2Y_1P_{28} + (\underline{\text{Fill}}\ \text{Code})IP_{31}$

Y_4: $\quad {}_1y_4 = U_2'U_1Y_5P_{25-0} + M_mT_1X_{11}x_7P_{28} + M_m(X_{11}' + T_1')Y_4'Y_3Y_2Y_1P_{28}$

$\quad {}_0y_4 = U_2'U_1Y_5'P_{25-0} + M_mT_1X_{11}x_7'P_{28} + M_m(X_{11}' + T_1')Y_4Y_3Y_2Y_1P_{28} + (\underline{\text{Fill}}\ \text{Code})IP_{31}$

Y_5: $\quad {}_1y_5 = U_2'U_1Y_6P_{25-0} + M_mT_1X_{11}x_8P_{28} + M_m(X_{11}' + T_1')Y_5'Y_4Y_3Y_2Y_1P_{28}$

$\quad {}_0y_5 = U_2'U_1Y_6'P_{25-0} + M_mT_1X_{11}x_8'P_{28} + M_m(X_{11}' + T_1')Y_5Y_4Y_3Y_2Y_1P_{28} + (\underline{\text{Fill}}\ \text{Code})IP_{31}$

Y_6: $\quad {}_1y_6 = U_2'U_1Y_7P_{25-0} + M_mT_1X_{11}x_9P_{28} + M_m(X_{11}' + T_1')Y_6'Y_5Y_4Y_3Y_2Y_1P_{28}$

$\quad {}_0y_6 = U_2'U_1Y_7'P_{25-0} + M_mT_1X_{11}x_9'P_{28} + M_m(X_{11}' + T_1')Y_6Y_5Y_4Y_3Y_2Y_1P_{28} + (\underline{\text{Fill}}\ \text{Code})IP_{31}$

Row Select Register (Con't)

Y_7:

$$1^{Y_7} = U_2'U_1WP_{25-0} + M_mT_1X_{11}X_{10}P_{28} + M_m(X_{11}' + T_1')Y_7'Y_6Y_5Y_4Y_3Y_2Y_1P_{28}$$

$$0^{Y_7} = U_2'U_1W'P_{25-0} + M_mT_1X_{11}X_{10}'P_{28} + M_m(X_{11}' + T_1')Y_7Y_6Y_5Y_4Y_3Y_2Y_1P_{28} + (\underline{\text{Fill}}\ \text{Code})IP_{31}$$

Operation Counter:

N_1:

$$1^{n_1} = GU_1(004 + 005)N_1'P_{31} + G_m(214)N_2P_{6-0} + G_mX_{27}N_1'P_1 + G_mX_{12}X_4P_{28}$$

$$0^{n_1} = GU_1(004 + 005)N_1P_{31} + GU_1'(004 + 005)P_{31} + G_m(214)N_2'P_{6-0} + G_mX_{27}N_1P_1$$
$$+ G_mX_{12}X_4'P_{28}$$

N_2:

$$1^{n_2} = GU_1(004 + 005)N_2'N_1'P_{31} + GU_1'(004 + 005)P_{31} + G_m(214)N_3P_{6-0} + G_mX_{27}N_2'N_1'P_1$$
$$+ G_mX_{12}X_5P_{28}$$

$$0^{n_2} = GU_1(004 + 005)N_2N_1'P_{31} + G_m(214)N_3'P_{6-0} + G_mX_{27}N_2N_1'P_1 + G_mX_{12}X_5'P_{28}$$

N_3:

$$1^{n_3} = GU_1'(004 + 005)P_{31} + G_m(214)N_4P_{6-0} + G_mX_{27}N_3'N_2'N_1'P_1 + G_mX_{12}X_6P_{28}$$

$$0^{n_3} = GU_1(004 + 005)N_2'N_1'P_{31} + G_m(214)N_4'P_{6-0} + G_mX_{27}N_3N_2'N_1'P_1 + G_mX_{12}X_6'P_{28}$$

Operation Counter: (Con't)

N_4: $${}_1n_4 = GU_1'(004 + 005)P_{31} + G_m(214)N_5P_{6-0} + G_mX_{27}N_4'N_3'N_2'N_1'P_1 + G_mX_{12}X_7P_{28}$$

$${}_0n_4 = GU_1(004 + 005)N_3'N_2'N_1'P_{31} + G_m(214)N_5'P_{6-0} + G_mX_{27}N_4N_3'N_2'N_1'P_1 + G_mX_{12}X_7'P_{28}$$

N_5: $${}_1n_5 = G_m(214)D_0P_{6-0} + G_mX_{27}N_5'N_4'N_3'N_2'N_1'P_1 + G_mX_{12}X_8P_{28}$$

$${}_0n_5 = G_m(214)D_0'P_{6-0} + G_mX_{27}N_5N_4'N_3'N_2'N_1'P_1 + G_mX_{12}X_8'P_{28}$$

Control Flip-Flops:

F_0: $${}_1f_0 = G_m(400)[(301)R_0 + (302)A_0 + (303)A_{31} + (304)D_{31} + (306)]P_0 + G_m(400)(305)KP_{31}$$

$${}_0f_0 = G_m(400)[(301)R_0' + (302)A_0' + (303)A_{31}' + (304)D_{31}' + (307)]P_0 + G_m(400)(305)K'P_{31}$$

F_1: $${}_1f_1 = G_m(401)[(301)R_0 + (302)A_0 + (303)A_{31} + (304)D_{31} + (306)]P_0 + G_m(401)(305)KP_{31}$$

$${}_0f_1 = G_m(401)[(301)R_0' + (302)A_0' + (303)A_{31}' + (304)D_{31}' + (307)]P_0 + G_m(401)(305)K'P_{31}$$

F_2: $${}_1f_2 = G_m(402)[(301)R_0 + (302)A_0 + (303)A_{31} + (304)D_{31} + (306)]P_0 + G_m(402)(305)KP_{31}$$

$${}_0f_2 = G_m(402)[(301)R_0' + (302)A_0' + (303)A_{31}' + (304)D_{31}' + (307)]P_0 + G_m(402)(305)K'P_{31}$$

Control Flip-Flops (Con't)

F_3: $\quad {}_1f_3 = G_m(403)[(301)R_0 + (302)A_0 + (303)A_{31} + (304)D_{31} + (306)]P_0 + G_m(403)(305)KP_{31}$

$\quad {}_0f_3 = G_m(403)[(301)R_0' + (302)A_0' + (303)A_{31}' + (304)D_{31}' + (307)]P_0 + G_m(403)(305)K'P_{31}$

State f.f.:

V_1: $\quad {}_1v_1 = (\underline{\text{Compute}}\ \text{Code})I_1(TR_b)(TP_b)P_{31} + (\underline{\text{Compute One-Cycle}}\ \text{Code})I_1(TR_b)(TP_b)P_{31}$

$\quad {}_0v_1 = GT_1'(009 + 012)P_{31} + V_2V_1U_2'U_1'P_0 + (\underline{\text{Fill}}\ \text{Code})IP_{31} + \text{Initial 0-Set}$

V_2: $\quad {}_1v_2 = (\underline{\text{Fill}}\ \text{Code})IP_{31} + G_1U_2P_{31}$

$\quad {}_0v_2 = V_2V_1U_2'U_1'P_0 + GT_1'(009 + 012)P_{31} + (\underline{\text{Compute}}\ \text{Code})I_1(TR_b)(TP_b)P_{31}$

$\qquad + (\underline{\text{Compute One-Cycle}}\ \text{Code})I_1(TR_b)(TP_b)P_{31} + \text{Initial 0-Set}$

Compute One-Cycle Control:

G_1: $\quad {}_1g_1 = (\underline{\text{Compute One-Cycle}}\ \text{Code})IP_{31}$

$\quad {}_0g_1 = (\underline{\text{Compute}}\ \text{Code})IP_{31}$

Input-Output Timing f.f.:

I_1: $\quad {}_1i_1 = [(TPP) + (TRP)]I_1'$

$\quad {}_0i_1 = [(TP_b) + (TR_b)]I_1 IP_{31}$

I: $\quad {}_1i = (\underline{\text{Allowable Input}}\ \text{Codes} + J)(TR_b)(TP_b)I_1 P_{31}$

$\quad {}_0i = IP_{31}$

Print Delay:

J: $\quad {}_1j = GU_1(006)P_{31}$

$\quad {}_0j = IP_{31}$

Input-Output Register:

H_1: $\quad {}_1h_1 = (\text{Channel 1}) + GU_1(006)H_2'P_{18-7} + V_2V_1'IH_2P_{31}'$

$\quad {}_0h_1 = GU_1(006)H_2'P_{18-7} + V_2V_1'IH_2' + IP_{31} + (TR_b)(\underline{\text{Non-Allowable}}\ \text{Codes})$

H_2: $\quad {}_1h_2 = (\text{Channel 2}) + GU_1(006)H_3'P_{18-7} + V_2V_1'IH_3P_{31}'$

$\quad {}_0h_2 = GU_1(006)H_3'P_{18-7} + V_2V_1'IH_3' + IP_{31} + (TR_b)(\underline{\text{Non-Allowable}}\ \text{Codes})$

Input-Output Register: (Con't)

H_3: $_1h_3 = (\text{Channel } 3) + GU_1(006)H_4P_{18-7} + V_2V_1'IH_4P_{31}'$

$_0h_3 = GU_1(006)H_4'P_{18-7} + V_2V_1'IH_4' + IP_{31} + (TR_b)(\underline{\text{Non-Allowable}}\ \text{Codes})$

H_4: $_1h_4 = (\text{Channel } 4) + GU_1(006)H_5P_{18-7} + V_2V_1'IA_0P_{31}'$

$_0h_4 = GU_1(006)H_5'P_{18-7} + V_2V_1'IA_0' + IP_{31} + (TR_b)(\underline{\text{Non-Allowable}}\ \text{Codes})$

H_5: $_1h_5 = (\text{Channel } 5) + GU_1(006)H_6P_{18-7}$

$_0h_5 = GU_1(006)H_6'P_{18-7} + IP_{31} + (TR_b)(\underline{\text{Non-Allowable}}\ \text{Codes})$

H_6: $_1h_6 = (\text{Channel } 6) + GU_1(006)M_0P_{18-7}$

$_0h_6 = GU_1(006)M_0'P_{18-7} + IP_{31} + (TR_b)(\underline{\text{Non-Allowable}}\ \text{Codes})$

Origin Pulse:

E_0: $_1e_0 = \text{Output of Origin Channel}$

$_0e_0 = E_0$

Pulse Counter:

E_1: $\quad {}_1e_1 = E_1 \cdot \overline{E_0}$

$\quad\quad {}_0e_1 = E_1 + E_0$

E_2: $\quad {}_1e_2 = E_2 \cdot \overline{E_1 E_0}$

$\quad\quad {}_0e_2 = E_2 E_1 + E_0$

E_3: $\quad {}_1e_3 = E_3 \cdot \overline{E_2 E_1 E_0}$

$\quad\quad {}_0e_3 = E_3 E_2 E_1 + E_0$

E_4: $\quad {}_1e_4 = E_4 \cdot \overline{E_3 E_2 E_1 E_0}$

$\quad\quad {}_0e_4 = E_4 E_3 E_2 E_1 + E_0$

E_5: $\quad {}_1e_5 = E_5 \cdot \overline{E_4 E_3 E_2 E_1 E_0}$

$\quad\quad {}_0e_5 = P_{31} + E_0$

Sum Driver:

$$A_s = (202 + 206)'(A_0'D_0'K + A_0'D_0K' + A_0D_0'K' + A_0D_0K)$$

$$+ (202 + 206)(A_1'D_0'K + A_1'D_0K' + A_1D_0'K' + A_1D_0K)$$

$$A_s' = (202 + 206)'(A_0D_0K' + A_0D_0'K + A_0'D_0K + A_0'D_0'K')$$

$$+ (202 + 206)(A_1D_0K' + A_1D_0'K + A_1'D_0K + A_1'D_0'K')$$

Command Execute Driver:

$$G = M_m'T_2'U_2$$

$$G' = M_m + T_2 + U_2'$$

Micro-Command Execute Driver:

$$G_m = M_mT_1U_1$$

$$G_m' = M_m' + T_1' + U_1'$$

Pulse Interval Drivers:

$$P_{31} = E_5E_4E_3E_2E_1$$

$$P_{31}' = E_5' + E_4' + E_3' + E_2' + E_1'$$

Pulse Interval Drivers: (Con't)

$$P_{30} = E_5 E_4 E_3 E_2 E_1'$$

$$P_{28} = E_5 E_4 E_3 E_2' E_1'$$

$$P_{16} = E_5 E_4' E_3' E_2' E_1'$$

$$P_7 = E_5' E_4' E_3 E_2 E_1$$

$$P_1 = E_5' E_4' E_3' E_2' E_1$$

$$P_0 = E_5' E_4' E_3' E_2' E_1'$$

$$P_{18-7} = E_5' E_4 + E_5 E_4' E_3' (E_2' + E_1') + P_7$$

$$P_{9-7} = E_5' E_4 E_3' E_2' + P_7$$

$$P_{12-7} = E_5' E_4 E_3' + E_5' E_4 E_2' E_1' + P_7$$

$$P_{27-0} = E_5' + E_4' + E_3'$$

Pulse Interval Drivers: (Con't)

$P_{25-0} = E_5' + E_4' + E_5E_4E_2'$

Miscellaneous Notations:

z_n = Core Read-Out from Column n.

x_n = Auxiliary Storage for Column n.

W = Output of Selected Memory Channel = $f_n(C_6 - C_1)M_{rn}$

TRP = Tape Reader Sprocket Signal

TPP = Tape Punch Sprocket Signal

TR_b = Back Contacts: Tape Reader

TP_b = Back Contacts: Tape Punch

(0xx) = $(S_4S_3S_2S_1)_{xx}$ = Command Code

(1xx) = $(Q_4Q_3Q_2Q_1)_{xx}$ = Arithmetic Micro-Command Code

(2xx) = $(S_4S_3S_2S_1)_{xx}$ = Control Micro-Command Code

VII CONCLUDING COMMENTS:

7.1 Possible Revisions:

The existing logical design could be simplified by deleting many of the micro-commands, eliminating some of the order types, removing the B Register, etc., but it is felt that this would greatly impair the usefulness of the machine. Although it is theoretically possible to perform any computation with a very restricted number of machine operations (Turing machine approach), this would not be done in practice.

Rather, the comments below describe possible changes in the micro-program facility which would enhance the versatility of the machine.

Enlarging the Core Memory:

Increasing the micro-order storage capacity of the core is probably the most useful modification. The need for a Type III order would be minimized, and, with sufficient core storage, it would be feasible to eliminate both the Type III and IV orders, incorporating the transfer of information from the drum to core as a Type I command.

A larger core facility causes negligible changes in the logical design and a small increase in the memory drive circuitry. The major cost would lie in the proportionate increase in core elements.

Expanding the Micro-Order:

Of the many possible modifications in the micro-order capabilities, those which appear to be the most useful are listed below.

a. The Arithmetic micro-commands which affect the end bits of the A, D, and R Registers could be expanded to include all possibilities.

This would decrease the need for recourse to the exchange micro-commands, shortening the micro-routines.

b. The Control micro-commands which cause transfer of the register contents could be expanded; in particular, A to D, and D to A transfer micro-commands would be convenient.

c. A useful modification would be to allow the Control Flip-Flops to sense any of the register end-bits.

d. The micro-routine control could be expanded by increasing the number of Control Flip-Flops, and by modifying the Y_n Control code to enable the programmer to select either the zero- or the one-state of a Control Flip-Flop for branching control (at present, the one-state causes the micro-routine to branch).

7.2 Tutorial Aspects:

An interesting consequence of the micro-program concept lies in the possible tutorial use of the machine. First of all, the derivation of a micro-routine bears resemblance in many aspects to the procedure followed in the logical design of conventional machines. Though the programmer is constrained by the micro-order structure of the machine, the "micro" scale of many of the operations requires consideration of digit operations, where the usual program concerns itself primarily with word operations.

Secondly, because of the micro-order structure of the machine, many of the Arithmetic and Control sequences are easily isolated; whether or not the logic is optimum, the student should have relatively less difficulty in understanding the methods by which common operations, such as addition, subtraction, shifting, complementing, etc. are performed.

7.3 Performing Logical Operations:

Aside from the use of digital computers as mathematical devices, there exists an extremely large area of utility where computers are used for non-arithmetic purposes. For example, a computer may be used to simulate a physical system, or may be programmed to reproduce the characteristics of some proposed special-purpose digital device and thus test the suitability of the design.

Even in the very fast scientific digital machines, however, it is difficult to achieve, for example, real-time characteristics in a simulation problem. This is often the result of an internal organization which requires equal time to perform a simple as well as a complex operation, or which must synthesize the simple operation from a series of more sophisticated commands. Although the problem has not been explored in detail, it appears that the micro-program concept is well-suited (perhaps best-suited) for such problems, where emphasis is placed upon the logical and decision-making capabilities of the machine.

7.4 Comparison with Conventional Serial Machines:

Except for the core memory, the proposed micro-programmed computer is estimated to correspond in its physical requirements to a conventional, serial general-purpose machine with perhaps 30 built-in commands, and with similar storage capacity and input-output equipment. The comparison below will assume such a machine.

To perform a particular operation, the micro-programmed machine would utilize a built-in command (only a limited selection available), a micro-routine, or a sub-routine containing combinations of micro-routines and

commands. The conventional computer uses either a built-in command or a sub-routine. For evaluation purposes, micro-routines in the proposed computer will be compared with built-in commands and with sub-routines containing commands.

Comparison Between Micro-Routines and Built-In Commands:

In general, a micro-routine is slower than a built-in command. This should be expected, since it would be assumed that built-in commands would be designed to be executed in the most efficient manner practical.

As an example, the micro-routines described in Part V require approximately 68 word-times for multiplication, and about 90 word-times for division. Built-in commands can perform these operations in 32 word-times (although there are machines which require 64 word-intervals).

Addition and subtraction are built-in operations for the micro-programmed machine, and are therefore executed efficiently.

Comparison Between Micro-Routines and Sub-Routines:

In a conventional computer, an operation not available in the command list will be performed by a sequence of commands, a sub-routine. A micro-routine is usually faster than the sub-routine required to achieve the equivalent result. This is true, not only for discretely different operations, such as square-root extraction, floating-point operations, etc., but also for variations of a basic command (these variations are commented on later).

For example, binary square-root extraction, as shown in Table 12, requires an average of about 170 word-times. A built-in command could perform this operation in less time, but a square-root command is a rarity. Usually, an iterative process is used, requiring several traversals of a sub-routine loop. The number of traversals is dependent upon

the initial guess for the root, and upon the convergence characteristics of the sub-routine; since multiplications and divisions are involved, the total time required is much more than 170 word-times.

Although many of the larger digital computers now incorporate floating-point commands, few of the smaller serial machines do. Feasible floating-point micro-routines have been derived, with approximate execution times as summarized below:

a. Floating-point multiplication: 108 word-times.

b. Floating-point division: 130-155 word-times.

c. Floating-point addition and subtraction: 70-190 word times, depending upon normalisation.

The floating-point micro-routines described operate on words which contain a signed characteristic and accompanying exponent. No supplementary memory access is required, so that floating-point operations are as natural in their use as are fixed-point commands.

In conventional machines, floating-point sub-routines often store the characteristic and exponent in separate words, require memory access to obtain orders and information during the sub-routine, and usually take longer than the intervals shown above.

Variations of a basic command can be included as part of a micro-routine, but require separate command sequences in the conventional computer. In a standard general-purpose machine, for example, double-length products, rounded products, etc. are either included as separate commands or require a sub-routine with auxiliary memory access. For the micro-programmed machine, the Control Flip-Flops can be used to branch into the micro-orders which perform the variations.

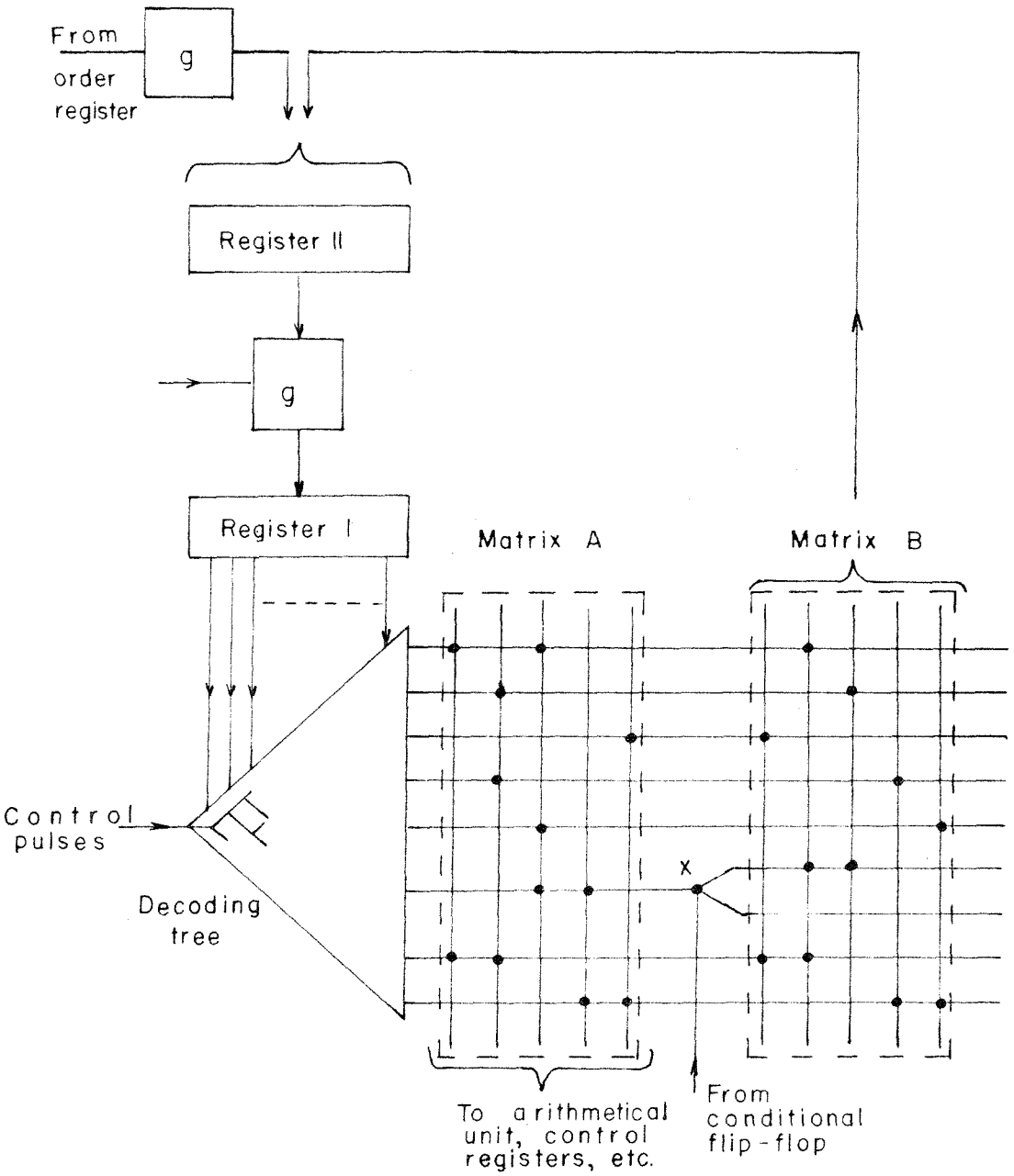
Conditional commands, most of which can be considered as variations

of a few basic operations, are readily synthesized by micro-routines.

Summary:

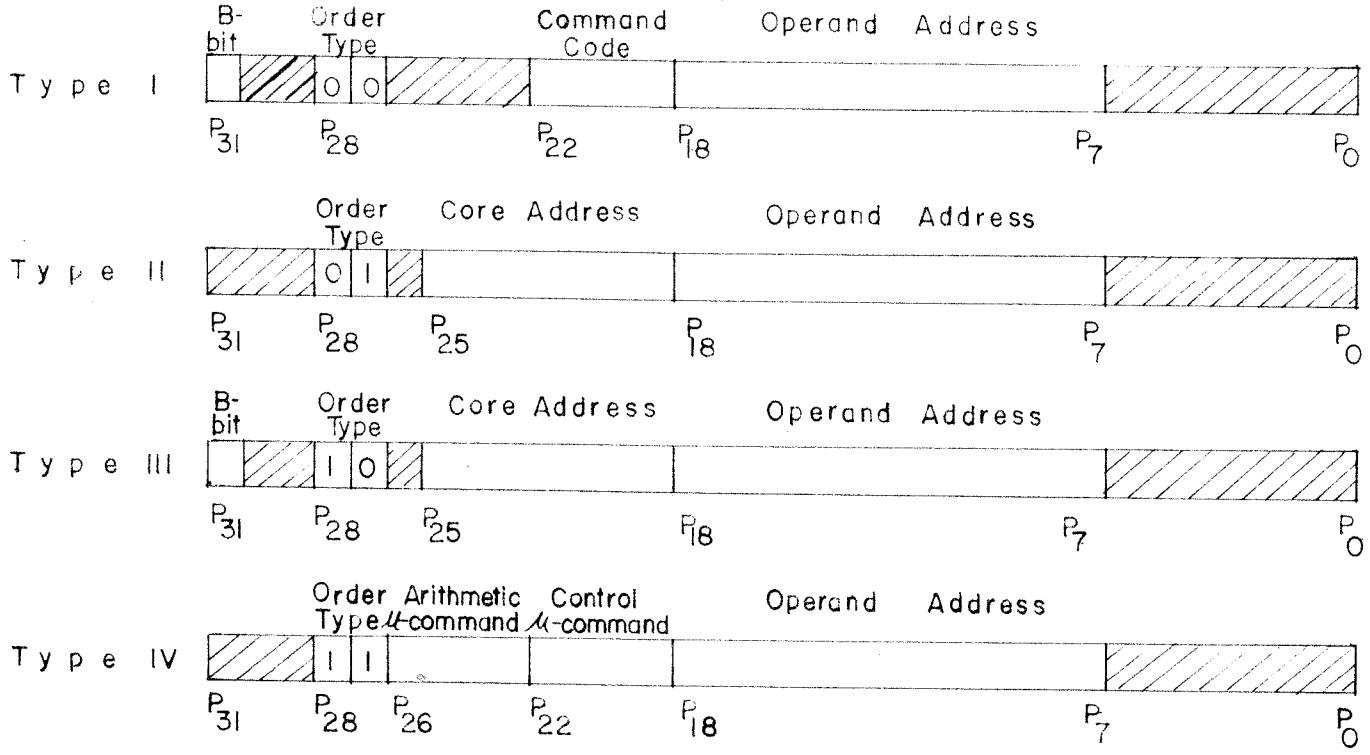
The core memory requirements of the micro-programmed computer are simple; furthermore, progress in this area and, in particular, the use of core elements for flip-flops and logic gates, would minimize the difficulty of incorporating core storage into the machine. Hence, the cost and complexity of the proposed machine is approximately equivalent to that of the previously mentioned conventional machine. Their general capabilities, however, differ, as noted below.

It appears that the proposed micro-programmed serial machine is best applied to non-repetitive problems or to problems where the programming effort consumes a major portion of the solution time. For often-used programs, or for problems where the computer performs related operations on a series of parameters, the faster command execution in a conventional computer should afford it the advantage, as long as the problem can efficiently be solved with the commands available.



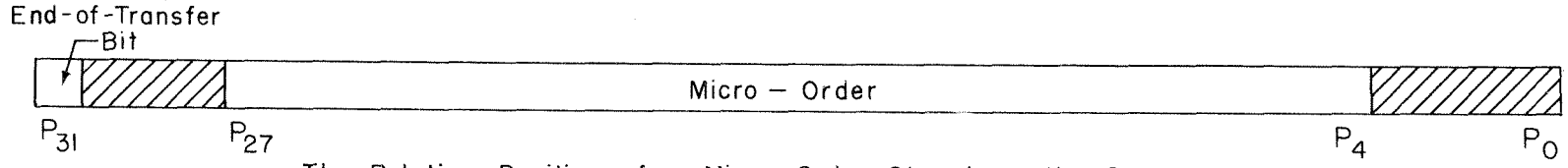
MICRO-CONTROL UNIT [FROM WILKES AND STRINGER (3)]

FIGURE 1

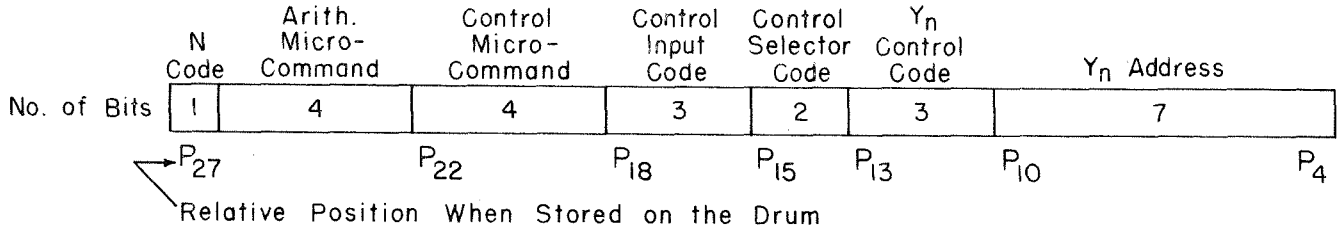


ORDER CONFIGURATION

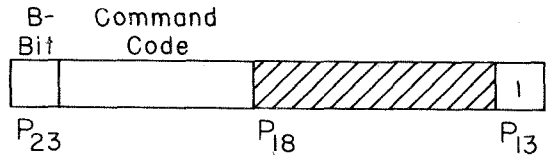
FIGURE 2



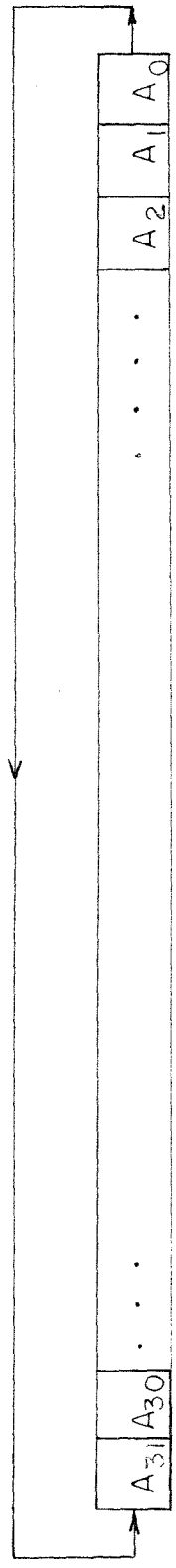
The Relative Position of a Micro-Order Stored on the Drum Memory



MICRO-ORDER CONFIGURATION
FIGURE 3

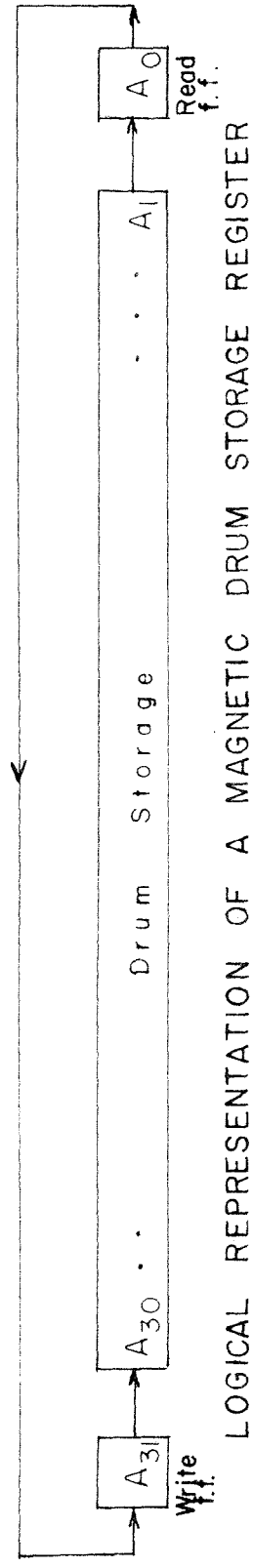


COMMAND STORED IN THE CORE MEMORY
FIGURE 4



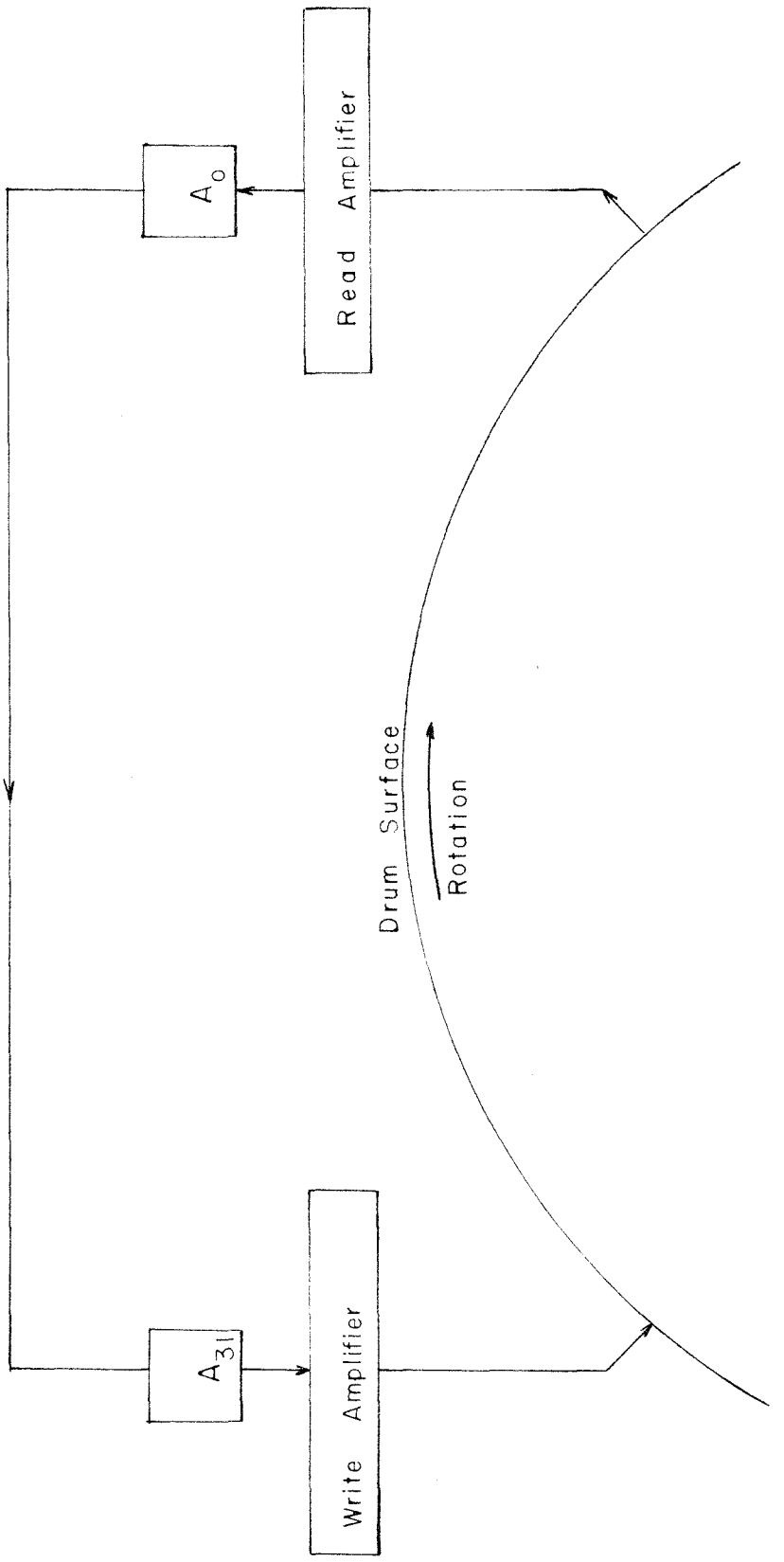
FLIP—FLOPS CONNECTED AS A DYNAMIC STORAGE REGISTER

FIGURE 5



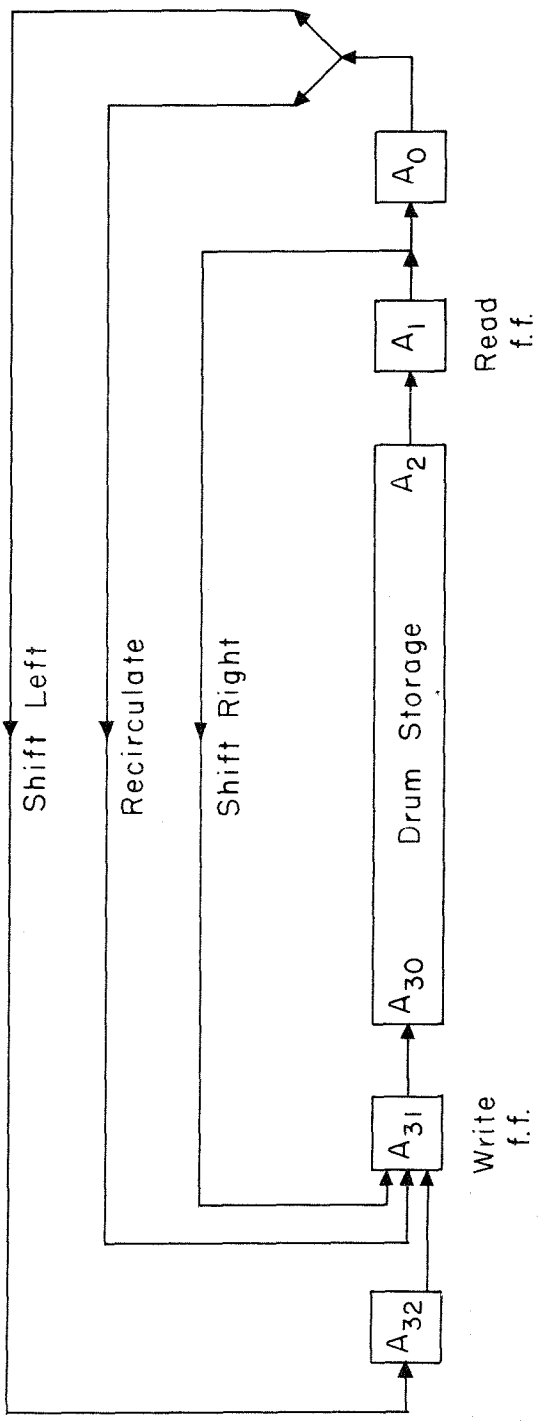
LOGICAL REPRESENTATION OF A MAGNETIC DRUM STORAGE REGISTER

FIGURE 6



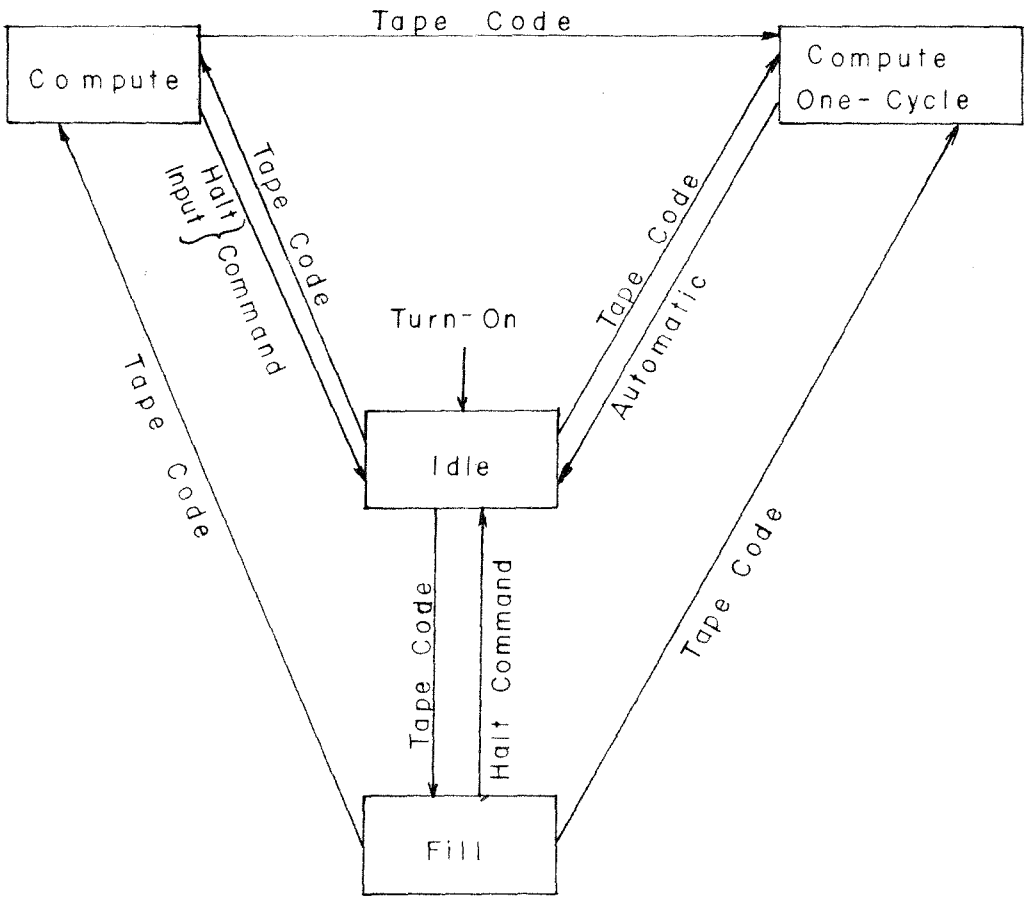
SCHEMATIC REPRESENTATION OF A MAGNETIC DRUM STORAGE REGISTER

FIGURE 7



STORAGE REGISTER CAPABLE OF SHIFTING IN EITHER DIRECTION

FIGURE 8



POSSIBLE TRANSITIONS AMONG INTERNAL STATES

FIGURE 9

TABLE 1

COMMAND LIST

000.	Null	
001.	Not Used	
002.	AM m.	A into C(m)
003.	MD m.	C(m) into D
004.	MS m.	Transfer to the Short Loop
005.	SM m.	Transfer from the Short Loop
006.	PT m.	Print
007.	PS m.	Partial Substitute
008.	EXT m.	Extract
009.	IN.	Input
010.	OCM m.	Store the Augmented Order Counter Contents
011.	MB m.	C(m) into B
012.	H.	Halt
013.	DB m.	Decrease B (Conditional Command)
014.	BA.	B into A
015.	IB.	Increase B

TABLE 2

MICRO-COMMAND LIST

Arithmetic Micro-Commands

100.	Null	
101.	comp A	Complement A
102.	a ADA	A + D into A
103.	s ADA	A - D into A
104.	a ADR	A + D into R
105.	s ADR	A - D into R
106.	cl A	Clear A
107.	cl R	Clear R
108.	cl D	Clear D
109.	0 msd D:	0 into msd D
110.	0 lsd R:	0 into lsd R
111.	1 lsd R:	1 into lsd R
112.	0 msd R:	0 into msd R
113.	1 msd R:	1 into msd R
114.	0 msd A:	0 into msd A
115.	1 msd A:	1 into msd A

Control Flip-Flop Inputs

300.	Null
301.	lsd R
302.	lsd A
303.	msd A
304.	msd D
305.	Carry at P_{31} (Set K)
306.	Set One
307.	Set Zero

Control Flip-Flop Selector

400.	F_0
401.	F_1
402.	F_2
403.	F_3

Control Micro-Commands

200.	Null	
201.	rmc	Return to Main Control
202.	sr A	Shift A Right
203.	sl A	Shift A Left
204.	src A	Circular Shift A Right
205.	slc A	Circular Shift A Left
206.	sr AR	Shift A and R Right
207.	sl AR	Shift A and R Left
208.	tr AMr	Transfer A to M Register
209.	tr AR	Transfer A to R
210.	tr RA	Transfer R to A
211.	ex AR	Exchange A and R
212.	ex AD	Exchange A and D
213.	ex RD	Exchange R and D
214.	tr DN	Transfer D into N Counter
215.	tr MrOC	Transfer M to Order Counter

Y_n Control Code

500.	Null	
501.	F_0	
502.	F_1	
503.	F_2	
504.	F_3	
505.	ut:	Unconditional
506.	tr Y_nN:	Transfer Y_n to the N Counter
507.	nmo:	Non-Micro-Order

TABLE 3

SOME RESULTS OF SIMULTANEOUS ARITHMETIC AND CONTROL MICRO-COMMANDS

Arithmetic Micro-Command	Control Micro-Command	Comment
a ADA s ADA	sr A	The shifted A is added* to D and placed into A.
a ADA s ADA	sr AR	Ditto above, but R is right-shifted, with original lsd A into msd R.
a ADR s ADR	sr A	The shifted A is added* to D, and placed into R. A has a normal right-shift.
a ADR s ADR	sr AR	Not permitted.
a ADA s ADA a ADR s ADR	sl A sl AR	The unshifted A is added* to D. The result is left-shifted. Other effects are analogous to the above.
a ADA s ADA	src A	Not permitted.
a ADR s ADR	src A	Adds* D to a word with all zeroes or all ones, depending upon original lsd of A.
a ADA s ADA	slc A	Not permitted.
a ADR s ADR	slc A	Normal sum* in R, but A has a circular shift.

* For a subtraction Micro-Command, make the obvious change.

TABLE 4

Order Execution Sequence

	Type I $T_2'T_1'$	Type II $T_2'T_1$	Type III T_2T_1'	Type IV $T_2'T_1'$
Phase I $U_2'U_1'$	Compares Order Counter and Sector Code. (W_cP_{3l} causes transition to Phase II.)			
Phase II $U_2'U_1$ (True for 1 word-time.)	Increase Order Counter by 1. Transfer Order Word to Address Register. Set T_2,T_1 with Order Type Code.			
	Store Command Code in S.	Put Micro-Routine start address into Row Select Register.	Put core fill address into Row Select Register.	Store Micro-Command codes into Q and S.
Phase III U_2U_1'	If commanded, B addition occurs during first cycle (O_c). If commanded, W_c compares Address Register to Sector code for operand pick-up.	Normally true for 1 word-time only. For command from core, return here from Phase IV.	If commanded, B addition occurs during first cycle (O_c). W_c compares Address Register to Sector Code for Micro-Routine pick-up.	skip
Phase IV U_2U_1	If required, O_c controls memory transfers. Execute the command.	Machine is in Micro-Mode. Read-out and execute Micro-Commands from core.	O_c causes pick-up from drum memory. Write information into core.	Executes Micro-Commands in Q and S.
Return to Phase I	Various "End-of-Command" signals.	"Return-to-Main-Control" micro-order.	"End-of-Transfer" code.	Automatic return after 1 word-time.

TABLE 5

Product And Quotient Sign Determination

Micro-Order No.	N Code	Arith Micro-Comm.	Control Micro-Comm.	Control Input	Control Selector	Y_n Control	Y_n Address
1				msd A	F_0	F_0	4
2				msd D	F_1	F_1	6
3				Set 0	F_0	ut	Out*
4		comp A		msd D	F_1	F_1	6
5				Set 1	F_0	ut	Out*
6			ex AD				
7		comp A					
8			ex AD			F_0	3
9				Set 1	F_0	ut	Out*

* "Out" refers to starting point of the subsequent micro-routine

TABLE 6

Multiplication

1		cl A	tr AR			tr Y_nN	30
2				lsd R	F_1	F_1	5
3	1		sr AR				2
4						ut	6
5	1	a ADA	sr AR				2
6			sr AR			F_0	8
7			rmc				
8		comp A	rmc				

TABLE 7

Modification for a Double-Length Product

Micro-Order No.	N Code	Arith Micro-Comm.	Control Micro-Comm.	Control Input	Control Selector	Y_n Control	Y_n Address
8			ex AR				
9		comp A		msd A	F_1	F_1	12
10		cl D	ex AR	msd A	F_1	F_1	13
11		comp A	rmc				
12		cl D	ex AR				
13		comp A					
14		1 lsd R	ex RD				
15			ex RD				
16		s ADA	rmc				

TABLE 8

Modification for a Single-Length Product With Round-Off

6		cl R	sr A	lsd A	F_1	F_1	10
7			ex DR				
8			a ADA			F_0	11
9			rmc				
10		1 lsd R				ut	7
11		comp A	rmc				

TABLE 9

Division

Micro-Order No.	N Code	Arith Micro-Comm.	Control Micro-Comm.	Control Input	Control Selector	Y_n Control	Y_n Address
1		s ADA		K	F_1		
2						F_1	4
3						ut	Out*
4			sl AR			tr $Y_n N$	32
5	1	a ADA		K	F_1		11
6		0 lsd R	sl AR			F_1	8
7						ut	5
8	1	s ADA		K	F_1		13
9		1 lsd R	sl AR			F_1	5
10						ut	8
11		s ADA					
12		0 lsd R	sl AR			ut	15
13		a ADA					
14		1 lsd R	sl AR				
15			ex AR			F_0	17
16			rmc				
17		comp A	rmc				

* Indicates overflow

TABLE 10

Conditional Command: Branch if A is Negative

Micro-Order No.	N Code	Arith Micro-Comm.	Control Micro-Comm.	Control Input	Control Selector	Y_n Control	Y_n Address
1				msd A	F_0	F_0	3
2			rmc				
3			tr MrOC			ut	2

TABLE 11

Conditional Command: Branch if A is Positive or Zero

1				msd A	F_0	F_0	3
2			tr MrOC				
3			rmc				

TABLE 12

Binary Square Root

Micro-Order No.	N Code	Arith Micro-Comm.	Control Micro-Comm.	Control Input	Control Selector	Y_n Control	Y_n Address
1		cl A	tr AD				
2		1 msd A				tr $Y_n N$	1
3	1		sr A				3
4			tr AR				
5			ex AD			tr $Y_n N$	15
6	1	s ADA		K	F_0		24
7		1 msd A		Set 1	F_1		
8			ex AD				
9			sr AR				
10			ex DR			F_1	18
11		s ADA	sl A			F_0	13
12		s ADA					
13			ex DR				
14			ex AD				
15			slc A			F_0	6
16	1	a ADA		K	F_0		26
17		0 msd A		Set 0	F_1	ut	8
18		a ADA	sl A			F_0	20
19		a ADA					
20			ex DR				
21			ex AD				
22			slc A			F_0	16
23						ut	6
24		a ADA				tr $Y_n N$	14
25		1 msd A				ut	28
26		s ADA				tr $Y_n N$	14
27		0 msd A					
28	1		slc A				28
29		0 msd A	rmc				

TABLE 13

Input: Fill Address Determination

Micro-Order No.	N Code	Arith Micro-Comm.	Control Micro-Comm.	Control Input	Control Selector	Y_n Control	Y_n Address
0				Set 0	F_0		
1		cl D	tr AR			tr Y_nN	1
2		cl A	rmc				
3		a ADA					
4		cl A	ex AD				
5		a ADA					
6		a ADA	sl A				
7			sl A				
8		a ADA	sl A				
9	1		ex AD				2
10						F_0	14
11				Set 1	F_0	tr Y_nN	5
12	1		sl A				12
13						ut	1
14			ex AR			tr Y_nN	6
15		a ADA					
16	1		sl A				16
17		cl A	tr AMr			ut	Out*

* To start of the micro-routine of Table 14

TABLE 14

Input: Word Interpretation

Micro-Order No.	N Code	Arith Micro-Comm.	Control Micro-Comm.	Control Input	Control Selector	Y_n Control	Y_n Address
1		cl R	sr A	lsd A	F_0		
2		cl D	sr A	lsd A	F_1	F_1	α_1
3			sr A	lsd A	F_1	F_1	α_2
4				lsd A	F_1	F_1	α_3
5						ut	α_4

TABLE 15

Input: Numerical Information (Decimal to Binary Conversion)

Micro-Order No.	N Code	Arith Micro-Comm.	Control Micro-Comm.	Control Input	Control Selector	Y_n Control	Y_n Address
1		1 lsd R					
2		cl A	rmc	Set 0	F_1		
3			src A			tr $Y_n N$	1
4			src A	lsd A	F_2	F_2	15
5	1		slc A				5
6		a ADA					
7		cl A	ex AD				
8		a ADA					
9		a ADA	sl A				
10			sl A				
11		a ADA	sl A			F_1	13
12		cl D	ex AR	Set 1	F_1	ut	6
13			ex DR				
14			ex AR			ut	2
15			src A			tr $Y_n N$	2
16				lsd A	F_2	F_2	18
17						ut	5
18			ex AD				
19		cl R	ex DR			ut	Out

TABLE 16

Input: Order Type Determination

Micro-Order No.	N Code	Arith Micro-Comm.	Control Micro-Comm.	Control Input	Control Selector	Y_n Control	Y_n Address
1		cl A				F_0	3
2			rmc			ut	5
3		l msd A					
4			slc A			ut	2
5			src A	lsd A	F_0	F_0	8
6			slc A			F_0	9
7		cl A	tr AR			ut	β_1
8				lsd A	F_0	ut	6
9		cl A	tr AR			ut	β_2

TABLE 17

Input: Type I, II, or III Order

Micro-Order No.	N Code	Arith Micro-Comm.	Control Micro-Comm.	Control Input	Control Selector	Y_n Control	Y_n Address
1				Set 0	F_1	tr Y_nN	2
2		cl D	rmc				
3		a ADA					
4		cl A	ex AD				
5		a ADA					
6		a ADA	sl A				
7			sl A				
8		a ADA	sl A				
9	1	cl A	ex AD				2
10						F_0	18
11			tr RA			tr Y_nN	7
12	1		sl A				12
13		a ADR				tr Y_nN	1
14		cl A				F_0	16
15				Set 1	F_0	ut	2
16						F_1	20
17				Set 1	F_1	ut	2
18			tr RA			tr Y_nN	5
19						ut	12
20			tr RA			tr Y_nN	6
21	1		sl A				21
22		a ADA				ut	Out

REFERENCES

1. EASIAC, A Pseudo-Computer, R. Perkins, Journal of the Assoc. for Computing Machinery, (April 1956), Vol. 3, pp. 65-72.

2. Preliminery Considerations on a Magnetic Drum Controlled Computer, E. F. Moore and T. Shapin, Jr., University of Illinois, Electronic Digital Computer, Internal Report No. 26 (Feb. 3, 1951).

3. Micro-Programming and the Design of the Control Circuits in an Electronic Digital Computer, M. V. Wilkes and J. B. Stringer, Proc. Cambridge Philosophical Society, (April 1953), Vol. 49, pp. 230-238.

4. Report of Manchester University Computer Inaugural Conference, July 1951 (Manchester 1953).

5. Mikroprogramm--Steuerwerk, H. Billing and W. Hopmann, Elektronische Rundschau, (October 1955), Vol. 9, pp. 349-353.

6. Use of Automatic Programming, W. F. Bauer, Computers and Automation, (Nov. 1956), Vol. 5, pp. 6-11, 36-37.

7. A Note on Microprogramming, H. T. Glantz, Journal of the Assoc. for Computing Machinery, (April 1956), Vol. 3, pp. 77-84.

8. Micro-Programming, R. J. Mercer, Journal of the Assoc. for Computing Machinery, (April 1957), Vol. 4, pp. 157-171.

9. IRE Standards on Electronic Computers: Definitions of Terms, 1956, Proc. IRE, (Sept. 1956), Vol. 4, pp. 1166-1173.

10. The Librascope General Purpose Computer, LGP-30, S. Frankel and J. Cass, Instruments and Automation, (Feb. 1956), Vol. 29, pp. 264-270.

11. Current Steering in Magnetic Circuits, J. A. Rajchman and H. D. Crane, IRE Trans. on Elec. Computers, (Mar. 1957), Vol. EC-6, pp. 21-30.

12. Magnetic Core Circuits for Digital Data-Processing Systems, D. Loev, W. Miehle, J. Paivenen, and J. Wylen, Proc. IRE, (Feb. 1956), Vol. 44, pp. 154-162.

13. Pulse Switching Circuits Using Magnetic Cores, M. Karnaugh, Proc. IRE, (May 1955), Vol. 43, pp. 570-584.

APPENDIX

Extracting a Binary Square-Root:

The general approach for extracting the square root of a binary number was first suggested to this writer by Dr. Stanley P. Frankel. Though the method is apparently well known, the specific results were not found in the literature inspected by this writer, and hence were derived.

Required: $\sqrt{a} = r$

Method: Let $a - r_i^2 \rightarrow 0$

where

$$r_{i+1} = r_i + \Delta r_i$$

and

$$\Delta r_{i+1} = \frac{1}{2}\Delta r_i$$

Therefore:

$$a - (r_{i+1})^2 = [a - (r_i)^2] - [(r_{i+1})^2 - (r_i)^2]$$

$$= [a - (r_i)^2] - 2r_i\Delta r_i - (\Delta r_i)^2$$

$$= [a - (r_i)^2] - C_i$$

If C_i is subtracted, and $a - (r_{i+1})^2 > 0$, then

$$r_{i+1} = r_i + \Delta r_i$$

and

$$C_{i+1} = \frac{1}{2}[C_i + \frac{3}{2}(\Delta r_i)^2]$$

so that,

$$a - (r_{i+2})^2 = [a - (r_{i+1})^2] - C_{i+1}$$

If C_i is subtracted, and $a - (r_{i+1})^2 < 0$,

then

$$r_{i+1} = r_i$$

and

$$C_{i+1} = \frac{1}{2}[C_i + \frac{1}{2}(\Delta r_i)^2]$$

so that,

$$a - (r_{i+2})^2 = [a - (r_{i+1})^2] + C_{i+1}$$

Extending the above results:

If C_i is added, and $a - (r_{i+1})^2 > 0$,

then

$$r_{i+1} = r_i + \Delta r_i$$

and

$$C_{i+1} = \frac{1}{2}[C_i - \frac{1}{2}(\Delta r_i)^2]$$

so that,

$$a - (r_{i+2})^2 = [a - (r_{i+1})^2] - C_{i+1}$$

If C_i is added, and $a - (r_{i+1})^2 < 0$,

then

$$r_{i+1} = r_i$$

and

$$C_{i+1} = \frac{1}{2}[C_i - \frac{3}{2}(\Delta r_i)^2]$$

so that,

$$a - (r_{i+2})^2 = [a - (r_{i+1})^2] + C_{i+1}$$