NOTES ON THE LOGIC OF THE ERA 1103 COMPUTER SYSTEM

20 JULY 1953

ERA 1103 COMPUTER LOGIC

TABLE OF CONTENTS

ERA 1103 COMPUTER LOGIC

1. FOREWORD

The word "logic", when applied to a large-scale digital computing system, means the set of rules by which the system operates. The following discussion will be concerned with those parts of the logic that are of importance to the programmer, namely, the set of instructions that he may give to the computer and the exact effect of these instructions on all the binary digits, or bits, that are stored in the computer. The more detailed parts of the logic, that determine how the effect of the instructions is achieved, will not be considered.

A comparison between the logic used in two different computing systems is not as easily made as a comparison between other important characteristics, such as reliability, word length, storage capacity, access time and addition time. The differences in logic may at first sight appear to be a number of subtleties that cannot be expected to have much practical value, yet the truth of the matter is that the differences between a good logic and an average one may well result in a very large improvement in overall performance.

The logic of the ERA 1103 is well balanced and flexible. Several unusual features combine to reduce both the number of instructions that have to be given to the computer by the programmer and also the number of separate operations that have to be performed by the computer. These features are general in nature, so that the improvements in performance will apply to a wide range of applications. Further the improvements in machine performance are not achieved at the expense of programming effort. On the contrary, the instructions are in many respects particularly convenient for the programmer, and they are not difficult to memorize.

The ERA 1103 computer is particularly well suited for real time applications, in which it is capable of acting as the central control and nerve center of a complex physical system. In such applications the flexible input-output system

will be one of the most important features of the computer. The 1103 is equipped with Magnetic Tape Storage, MT, that can provide auxiliary storage for problems that involve the handling of very large quantities of information. The present discussion, however, is mainly concerned with the operations that are carried out in the arithmetic section of the machine and in the computing memory that is composed of electrostatic storage, ES, and magnetic drum storage, MD; it is here that the actual computation is done. The aim is to give an account of the computer instructions of the 1103 in sufficient detail to answer most of the first round of questions that a programmer would ask.

Before discussing the instructions it is necessary to draw attention to the overall logic of the computer, which differs from most other computers in two important respects. The first of these differences is the integration of the magnetic drum and electrostatic memories with the arithmetic registers in one system of memory box addresses; the second is the provision for the accumulation of double-length numbers. In the subsequent discussion of the instructions the most startling feature is the "Repeat", but there are several other important novelties, and almost all the instructions are of some interest to a programmer because they indicate the efficiency of a two-address logic.

2.0 INTRODUCTION TO CHARACTERISTICS

2.1 TERMINOLOGY

The terminology used in discussing the characteristics of the 1103 computing system is explained in the following notes.

1. Word length

   36 bits (binary digits)

2. Parallel access registers of Arithmetic Section

   A 72-bit accumulator (a_{71}, a_{70}, ..., a_0)

   Q 36-bit shifting register (q_{35}, q_{34}, ..., q_0)

   X 36-bit exchange register (x_{35}, x_{34}, ..., x_0)

   A_R right-hand (least significant) 36 bits of A.

   A_L left-hand (most significant) 36 bits of A.

3. Parallel access storage

   ES 1,024 words of Electrostatic Storage

   MD 16,384 words of Magnetic Drum Storage

4. Composition of an instruction word.

Instruction word	36 bits (i_{35}, i_{34}, ..., i_0)
Operation code	6 bits (i_{35}, i_{34}, ..., i_{30})
First execution address, u,	15 bits (i_{29}, i_{28}, ..., i_{15})
Second execution address, v,	15 bits (i_{14}, i_{13}, ..., i_0)

5. Allocation of addresses

ES	00000 - 01777	(octal)
Q	1----	(octal)
A	2----	(octal)
MD	40000 - 77777	(octal)

6. Sections of addresses

   j one digit octal number represented by $u_{14}u_{13}u_{12}$.

n four digit octal number represented by $u_{11}, u_{10}, \ldots, u_0$.

k number of shifts, represented by $v_6, v_5, \ldots, u_0$.

7. Contents of registers and memory boxes

A prime is a complement, such as: (Q') = complement of (Q)

Parentheses are used to denote "contents of". Thus:

(u) = 36-bit word at address u.

(Q) = 36-bit word in Q.

(A) = 72-bit word in A.

(A_R) = 36-bit word in A_R.

(A_L) = 36-bit word in A_L.

8. Double-length extensions

D(u) = 72-bit word whose right-hand 36 bits are (u) and whose left-hand 36 bits are all alike and equal to the left-most bit of (u).

S(u) = 72-bit word whose right-hand 36 bits are (u) and whose left-hand 36 bits are all zero.

D(Q), D(X), S(Q), and S(X) are similarly defined.

L(Q)(u) = 72-bit word whose left-hand 36 bits are zeros and each of whose right-hand 36 bits is given by the product of the corresponding bits of (u) and (Q).

L(Q')(v) = 72-bit word whose left-hand 36 bits are zeros and each of whose right-hand 36 bits is given by the product of the corresponding bits of (v) and the complement of (Q).

9. Control registers

PAK Program Address Counter

UAK U-Address Counter

VAK V-Address Counter

SAR Storage Address Register

Note that (PAK), (UAK), (VAK), (SAR) denote the contents of these registers.

10. Arrangement of MD: Interlace

4,096 bits on each track

4,096 words on a group of 36 tracks

4 groups of tracks, giving 16,384 words

Variable interlace between the Storage Address Register and the angular location counter permits choice of the angular interval between memory locations with consecutive addresses.

11. Input-output registers

IOA An in-out register of 8 bits.

IOB An in-out register of 36 bits.

HPR A high-speed punch register of 6 bits

TWR A typewriter register of 6 bits.

12. Sequence of instructions

PI Previous instruction

CI Current instruction

NI Next instruction

13. Program sequence control - Jumps

A complete computer operation consists of two parts

Part One - the execution of CI

Part Two - the acquisition of NI

At the start of Part One the program control registers already contain CI, as the result of the second part of the previous operation, and (PAK) is x + 1, where x is the address from which CI was acquired.

During Part Two NI is acquired from address held in PAK at the end of Part One, and (PAK) is then increased by one.

Thus, provided that CI does not call for a change in (PAK), NI will be acquired from address x + 1. In a normal program sequence, successive instructions are obtained from consecutive addresses.

A departure from the normal sequence is called a "jump", and is achieved by altering (PAK) during Part One of an operation. Instructions that call for a change in (PAK) are called "jump" instructions.

2.2 THE COMPUTING MEMORY

An unusual and important feature of the 1103 is the fact that neither electrostatic storage, ES, nor the magnetic drum, MD, is the primary storage of the computer. Instead the two classes of storage are closely integrated to form a single computing memory of large capacity, and the association of ES with MD makes the latter class of storage far more valuable than it would be by itself. The programmer has considerable freedom to vary his methods of using ES and MD in order to suit the peculiarities of different problems.

It will be noted in 2.1 (5) above that the left-most bit of a 15-bit address determines whether the address is in MD or not. When this bit is zero, the first octal digit of the address distinguishes between ES, Q, and A. The ES and MD address ranges are treated by PAK, UAK, and VAK as closed consecutive sets. The control of the four classes of storage is facilitated by the use of an exchange register (X-register) as a switching center for most of the internal transmissions, including the transmission of instructions from storage to the control registers. This register, X, has sign sensing and complementing facilities; it serves as a buffer in transmissions between the several classes of storage, including the A and Q registers.

The inclusion of A and Q in the address system is a convenience for the programmer, but a feature of far greater importance is the fact that the individual memory boxes of MD, as well as those of ES, have addresses. In other computers using MD and ES, the former is treated in much the same way as the Magnetic Tape Storage (MT) is treated in the 1103. (In the 1103, Magnetic Tape Storage is supplied as an integral part of the computer. Four magnetic tape units are provided. The tape information is arranged in blocks of 32 words each. Transmission of information to and from MT is by blocks, and is executed automatically in response to special instructions.) In these other computers,

access to individual memory boxes in MD can only be obtained by means of transfers between MD and ES: program instructions cannot be directly acquired from MD, but must be transmitted to ES in blocks for execution. Because of these restrictions the computing memory, in which all instructions and quantities must be held when they are required, is limited in these other computers to ES and the arithmetic registers. In the 1103 the computing memory includes the 16,384 boxes of MD as well.

When programming for the 1103 it is possible to transfer blocks of instructions from MD to ES prior to execution, and, as will be seen later, there are particularly efficient instructions for this purpose. It is important to note, however, that the programmer also has the option of keeping the program in MD for execution, and that in many cases this gives him the equivalent of over 17,000 registers of fast-access storage. With this large computing memory at his disposal he can expect to write programs that will require relatively few separate computer operations, and will, therefore, give high computing speeds.

2.3 ARITHMETIC IN THE 1103

An outstanding feature of the arithmetic in the 1103 is the existence of a single instruction, called Multiply Add (MA), that causes the 72-bit product of the two 36-bit numbers at addresses u and v to be added to the 72-bit content of the accumulator. The value of this instruction in forming a scalar product

$$a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$$

is evident; it is merely necessary to repeat the MA instruction n times with appropriate u and v-addresses, and the scalar product will be built up in the accumulator, A, provided that A was cleared initially.

In the 1103 the accumulator, A, is a genuine double-length accumulator, holding a 72-bit word. For arithmetic operations this word is regarded as an integer, the binary point being taken to be at the right-hand end. The left-most bit, a_{71}, is a sign-bit. Thus zero and the positive integers up to $2^{71}-1$ can be represented by words in A whose sign-bit is zero. When the sign-bit is one, (A) is taken to represent the negative of the integer that is obtained by complementing all the bits of (A), including the sign bit. In other words, whether (A) represents a positive or a negative number, the negative of (A) is obtained by complementing all the individual bits of (A). The full range of numbers that can be represented in A is from $1 - 2^{71}$ to $2^{71} - 1$.

Numbers added or subtracted into the accumulator are always 72-bit numbers. Such numbers are transmitted to A from the exchange register X, which holds only a single-length word of 36 bits. The usual procedure amounts to forming the double-length extension D(X) of (X), as defined in 2.1(8) above, and then adding D(X) to (A). It should be noted that the addition of D(X) to (A) is actually performed by subtracting the complement of D(X). On the other hand subtraction of D(X) from (A) is done directly, without complementing D(X)

The fact that the basic operation of the 1103 accumulator is subtraction rather than addition will not affect the programmer, except that any zero that may appear in A as the result of an arithmetic operation will be represented by 72 zero bits, rather than by 72 ones, as would happen if the accumulator were basically additive.

Numbers stored in Q and in single memory boxes of ES and MD will be 36-bit numbers. Again the binary point is taken to be at the right-hand end, the left-most bit is a sign-bit, and the one's complement representation of negative numbers is used. Thus, the single-length numbers are integers ranging from $1 - 2^{35}$ to $2^{35} - 1$.

The one's complement number system in an additive accumulator requires a correction, that is usually known as "end around carry". Since the 1103 has a subtractive accumulator, the corresponding correction takes the form of "end around borrow". Any borrow that may occur to the left of the extreme left-hand place, a_{71}, is applied at the extreme right-most place, a_0.

In some situations it is necessary to handle quantities in A as single-length quantities. For this purpose, as will be seen later, special "split" instructions are provided. In these instructions, and in a few other operations, the "split extension" S(X) is used, as defined in 2.1(8) above, instead of the "double-length extension" D(X).

All arithmetic operations involving (A) are achieved by a combination of the following four elementary operations, together with transmissions to or from X.

a) Clearing A.

b) Subtracting D(X) or S(X) or the complement of one of these 72-bit numbers from (A).

c) Transmitting (A_R) to X without changing (A).

d) Shifting (A) to the left in a circular manner.

Although the 1103 treats numbers as integers, the programmer can, of course, introduce scale factors that will enable him to handle numbers of any size. Further he can program for floating point operation where this is necessary. Many programmers will be more familiar with what may be called a "fractional system of arithmetic". In a computer that uses the fractional system, numbers are taken to lie between -1 and +1, the binary point being immediately to the right of the left-most bit, which is used as a sign-bit. In such a computer the accumulator may be of double-length, but single length words are added into the left-hand end. This means that the programmer must be sure that the sum of two numbers is less than one in magnitude before he can instruct the computer to add them. In the 1103, on the other hand, the programmer can safely call for the addition of any two single-length numbers without preliminary scaling, since the single-length numbers are effectively added into A_R and the whole of A_L is available to hold any overflow of the capacity of A_R.

The arithmetic system of the 1103, which incidentally was also used in the 1101, has several advantages over the fractional system and calls for a different attitude to overflow. In fractional arithmetic the accumulator will contain a false result if an addition or subtraction is allowed to exceed the single-length capacity. Consequently, computers using the fractional system are apt to have built in circuitry that will give an alarm signal whenever an overflow of the single-length capacity occurs. On the 1103, with a single-length capacity of 36 bits, the accumulator will not contain a false result unless the double-length capacity of 72 bits is exceeded. The programmer will want to take advantage of this feature, and hence will not have need of an alarm for a single-length overflow.

Suppose for example that a programmer wants to obtain the sum

$$a_1 + a_2 + \ldots + a_n$$

of a set of single-length, or 36-bit, quantities a_1, a_2, ... a_n. In order to retain the greatest possible accuracy, he will want these quantities to contain as many significant digits as possible, so he would like to scale them up as near as possible to the full capacity of a single-length number. If he were using a computer with a fractional system, he would have to scale down before adding, which means a loss of significant digits. If he is using the 1103, he will not need to scale down until he wants to transmit the sum to storage as a single-length word. He can, therefore, scale down after the summation rather than before, which gives greater accuracy.

Another unusual feature of the 1103 arithmetic is the advantage one may take of the double-length accumulator when a summation is followed by a division. In the divide operation, the dividend will be the entire 72-bit content of the accumulator.

When a programmer is working with the accumulated sum of a set of 36-bit numbers, he is likely to need a test for overflow beyond a prescribed place in A_L. Such a test can easily be programmed. A different situation arises when a programmer is taking advantage of the Multiply Add instruction to form a 72-bit scalar product. In this case he will be working with the accumulated sum of a set of 72-bit numbers, and he will, of course, try to ensure that the full 72-bit capacity of A is not exceeded. As a safeguard, however, an overflow alarm is provided that will cause the computer to stop whenever an overflow of the 72-bit capacity is imminent.

Shifting facilities in both Q and A are based on circular shifts to the left, in which the bit that is pushed off the left-hand end is inserted at the right-hand end. Open-ended shifts and round-off are handled on a subroutine basis, such operations being called for by a single "Return Jump" (RJ) instruction

that causes the program control to "jump" to the appropriate subroutine and also arranges for the return jump. More will be said about this later.

2.4 OPERATION TIMING

Each instruction is executed by a sequence of basic commands, each of which represents a single pulse sent out over a corresponding line. This sequence is generated by control pulses derived from internal clock pulses. The 500 kc clock pulses are derived from 125 kc timing pulses recorded on a channel of the magnetic drum.

Clock pulses are fed into a Main Pulse Distributor, which distributes them as Main Control Pulses to eight lines in sequence. For an individual command, which may require less than eight successive control pulses for its execution, certain of the eight lines are skipped in this sequence to provide a correspondingly shorter pulse cycle.

This main pulse sequence can be halted by the initiation of certain commands which give rise to a sequence of subsequent commands under the jurisdiction of a secondary control. The secondary controls consist of Storage Class Control, Arithmetic Sequence Control, Repeat Sequence Control, Magnetic Tape Access Control, and Input-Output Control. In general, the main sequence is halted until a resume pulse indicates that the secondary control has performed its called-for function; the Repeat Sequence, however, uses the main pulses to execute the repetition of instructions, while the Input-Output Control proceeds independently unless a second input-output operation is called for before a prior one has been completed.

The Storage Class Control selects and initiates the sequence of commands required for a reference to storage, the particular sequence being dependent on which class of storage is referred to. Reference to the accumulator or to the Q-register involves fixed sets of pulses with no interruption of control pulses. On the other hand, reference to the magnetic drum involves interruption of pulses until the drum has reached the proper angular position, while

reference to the electrostatic storage involves interruption until the proper point in the electrostatic storage cycle is reached.

The electrostatic cycle is generated by a continual cyclic distribution of clock pulses to four lines in sequence. The commands initiated by these four pulses are summarized in the following table.

Electrostatic Pulse	Commands For Regenerate	Read	Write
0	Position beam to contents of Regeneration Counter (RK); advance RK.	Position beam to contents of SAR.	Position beam to contents of SAR;
1	Wait	Wait	Wait
2	Probe Sample stored digit	Probe Sample stored digit	Probe
3	Restore digit if a one	Restore digit if a one, Read out Clear SAR Resume main pulse	Write digit Clear SAR Resume main pulse

When a storage reference to ES is made, the next electrostatic pulse 0 is awaited before the cycle is put into the read or the write mode. After reading or writing, the cycle returns automatically to the regenerate mode.

The Arithmetic Sequence Control selects and initiates the sequence of commands required for the sequenced operations within the arithmetic section. Some of these operations may be of variable length depending upon the contents of arithmetic registers (for example, Scale Factor, Multiply, and Divide).

Because of the above indicated variable execution times of certain commands initiating secondary control commands, the execution times of an individual instruction has to be specified for each of several cases. For example, the instruction Transmit Positive u v requires four main pulses for execution:

Main Pulse	Commands
0	Transmit (UAK) to SAR: Initiate Read
5	Transmit (VAK) to SAR: Initiate Write
6	Transmit (PAK) to SAR: Advance PAK: Clear X
	Clear Program Control Register; Initiate Read
7	Transmit (X) to Program Control Register: Clear X

The Main Pulse Distributor is not allowed to advance after pulses 0, 5 and 6 until a storage resume has been initiated. One clock pulse period passes between the time that the Main Pulse Distributor was last advanced and the beginning of a read or write cycle. The total times for the read and write cycles will vary according to the classes of storage to which u and v refer. Since one may not always begin the reading or writing at the most advantageous time, there may be a variation in the total time for the read or write cycle. Assuming that the instructions are being obtained from ES, the following reference intervals (in clock pulse periods) apply for the various address class combinations.

	u class -	ES	ES	ES	A	A	Q	Q
Interval	v class -	ES	A	Q	ES	Q	ES	A
0-5		5	5	5	1	1	2	2
5-6		7	4	2	7	2	6	4
6-7		7	6	8	7	8	7	5

There have been a total of three clock pulse periods (MP0, MP5 and MP6) initiating the reading or writing from a given storage class. One clock pulse period is used to carry out main pulse 7 and an additional clock pulse is used to allow the output of the Main Control Translator to become established before main pulse 0 of the next instruction cycle. Thus, five pulse periods are involved in the main sequence portion of this instruction execution. The total execution time then becomes for the different cases of the Transmit Positive u v instruction:

24 20 20 20 16 20 16

When u and/or v refer to MD, other times can arise. For these cases, a storage reference to MD can require a minimum of one pulse period or a maximum of one drum revolution time (34 msec). Consecutive storage references (read or write) to MD can always be made at an interval of 16 + 4c pulse periods ($c \geq 0$), which corresponds to an interval of 4 + c drum cell periods.

When instructions are being executed from MD, or when an instruction with one or both operands in MD is repeated by the use of the Repeat instruction, consecutive references to MD are made at consecutive addresses. In order that a drum revolution interval be avoided, it is necessary that cells having consecutive addresses be a sufficient number of cells apart. This is made possible by allowing the correspondence between the Drum Angular Position Locating Register, and the Storage Address Register to be selectable. The standard interlaces are obtained by taking $(SAR)_{13}$ $(SAR)_{12}$ as the Group Selector digits, and wiring $(SAR)_i$ to the same Coincidence Detector stage as $(LR)_j$ where $0 \leq i \leq 11$ and $j = i + k \bmod 12$ for some fixed k. The resulting interlace is termed a 2^k-interlace and assigns consecutive addresses to cells 2^k cell-intervals apart. The smallest useful value of k is 2, corresponding to a 4-interlace. For any particular computer program , the programmer selects that interlace which is suitable for the entire program.

3.0 EXPLANATION OF INSTRUCTIONS

3.1 METHOD OF PRESENTATION

The instructions can be more readily memorized if they are considered in groups. In the following description the sequenced instructions are taken first; these are the most complex instructions and must be memorized individually. Then the basic arithmetic and manipulation instructions are arranged in four groups, in accordance with the use that is made of the execution addresses u and v. Then the eight jump instructions are arranged in three groups. Finally, the instructions affecting MT, input and output are briefly described.

Programmers who intend to do a great deal of work with the 1103 will memorize the pairs of octal digits that represent the operation sections of the instructions, but a two-letter representation is easier to learn and is, therefore, more convenient for a discussion of the computer or for those who wish to write only a few programs. Unfortunately, a two-letter code is bound to suffer from the following difficulties:

(a) A stands for add, address, accumulator, advance.

(b) D stands for divide, double-length.

(c) M stands for multiply, manual, mask.

(d) S stands for subtract, shift, scale, split, sign, select, stop.

(e) L stands for left, logical

It is, therefore, almost inevitable that some of the abbreviations used for the 1103 instructions will have been used with other meanings for other computers.

3.2 SEQUENCED INSTRUCTIONS

The following five instructions involve complex built-in sequence control:

MPuv — MultiPly: Form in A the 72-bit product of (u) and (v), leaving in Q the multiplier, (u).

MAuv — Multiply Add: Add to (A) the 72-bit product of (u) and (v), leaving in Q the multiplier (u).

DVuv — DiVide: Divide the 72-bit number (A) by (u), putting the quotient in Q, and leaving in A a non-negative remainder, R. Then replace (v) by (Q). The quotient and remainder are defined by: $(A)_i = (u) \cdot (Q) + R$ where $0 \leq R < |(u)|$. Here $(A)_i$ denotes the initial contents of A.

SFuv — Scale Factor: Replace (A) with D(u). Then left circular shift (A) by 36 places. Then continue to shift (A) until $A_{34} \neq A_{35}$. Then replace the right-hand 15 bits of (v) with the number of shifts, k, which would be necessary to return (A) to its original position. If (A) is all ones or zeros, k = 37. If u is a, (A) is left unchanged in the first step, instead of being replaced by $D(A_R)$.

RPjnw — RePeat: This instruction calls for the next instruction. which will be called NIuv, to be repeated n times, its "u" and "v" addresses being modified or not according to the value of j. Afterwards the program is continued by the execution of the instruction stored at a fixed address F_1. Normally F_1 is the ES address (00000), but it can be changed to the MD address (40001) by means of a relatively inaccessible switch. The exact steps to be carried out are:

(1) Replace the right-hand 15 bits of (F_1) with the address w.

(2) Execute NIuv, the next instruction in the program, n times.

(3) If j = 0 do not change u and v.

If j = 1 add one to v after each execution.

If j = 2 add one to u after each execution.

If j = 3 add one to u and v after each execution.

(4) On completing n executions, take (F_1) as the next instruction.

(5) If the repeated instruction is a jump instruction, the occurrence of a jump terminates the repetition. In addition, if NIuv is a Threshold Jump or an Equality Jump, and the jump to address v occurs, (Q) is replaced by the quantity j, (n - r), where r is the number of executions that have taken place.

To illustrate the effectiveness of the RP instruction, consider the scalar product:

$$a_0 b_0 + a_1 b_1 + a_2 b_2 + \ldots + a_{n-1} b_{n-1}$$

and suppose that the quantities a_i and b_i are 36-bit numbers stored at two sets of consecutive memory boxes with addresses u + i and v + i, where i = 0, 1, 2, . . ., (n-1). If A is cleared initially, the scalar product can be obtained as a 72-bit number in A by use of the two instructions:

RP3nw

MAuv

The memory box F will contain an unconditional jump instruction whose v-address will be set to w by the RP instruction, so the program goes to the instruction

in address w after the scalar product has been obtained.

The time required to obtain the 72-bit scalar product will depend on the number of ones in the multipliers a_0, a_1, . . . a_{n-1}. Taking an average of 166 clock pulses for each execution of MA, the total time requirement, including the execution both of the RP and of the jump to address w, is 46 + 166n clock pulses.

This method of obtaining a scalar product is remarkable in three ways:

(1) As it uses only two instructions it is extremely economical of storage space.

(2) It is fast because the MA instruction is acquired by only one reference to storage although it is used repeatedly.

(3) It obtains the scalar product as a double-length 72-bit number. (On other computers the two 36-bit halves of each product a_i b_i would have to be handled separately, which involves several instructions and several references to storage for each step of the accumulation of a 72-bit scalar product.)

The effectiveness of the combination of RP and MA is largely responsible for the remarkable efficiency of the 1103 computer in handling matrix multiplications. Two 16 by 16 matrixes can be stored in ES and can be multiplied in 1.7 secs. Two 64 by 64 matrixes can be stored in MD and can be multiplied in 4.5 minutes. Larger matrixes can be handled by using MT.

The above scalar product procedure may be used to perform decimal-to-binary conversion. Suppose that N is a positive integer not greater than 3×10^{10} and that a_0. a_1. a_2. . . . ., a_{n-1} are the individual decimal digits of the decimal form of N. Then, if $b_i = 10^i$, the two instructions RP3nw and MAuv will evaluate the expression:

$$a_0 + 10a_1 + 10^2 a_2 + \ldots + 10^a a_{n-1}$$

and will, therefore, obtain N in binary form in A_R. For 11-digit decimal integers not greater than 3×10^{10} the time taken to perform this conversion is about 2.8 milliseconds.

Equally remarkable is the fact that the reverse conversion, from binary to decimal, can also be achieved by only two instructions, namely,

RP3nw

DVuv

In this case the positive integer N in binary form is placed in A and $a_i - 10^{n-i}$. The number n is the number of decimal digits required in the decimal form of N, and these digits are obtained successively in binary-decimal form at addresses v + i, where i - 0, 1, 2, . . . (n-1). For n = 10, the time required for the conversion is about 5.4 milliseconds.

It has already been pointed out that the DV instruction has the unusual feature of being able to handle a 72-bit dividend. An alarm is provided that will stop the computer if the correct quotient of a division would exceed the 36-bit capacity of a single-length register.

The Scale Factor instruction is very important for floating point operation and for many cases of function evaluation in which preliminary scaling is desirable. No special comment is required here because similar instructions have been used in other computers.

In the sequenced instructions the only pecularities that arise from the use of q and a as execution addresses are as follows:

Instruction	Quantity left in A
MPua	0 (all zeros)
MPqa	0
MPaa	0

Instruction	Quantity left in A
MPuq	$(u)^2$
MPaq	$(A_R)^2$
MAua	$(A) + (u)(A_L)$
MAqa	$(A) + (Q)(A_L)$
MAaa	$(A) + (A_R)(A_L)$
MAuq	$(A) + (u)^2$
MAaq	$(A) + (A_R)^2$

In the above list the address u is assumed to be either in ES or in MD.

Before leaving the sequenced instructions a further remark on the program sequence control associated with the RP instruction is needed. During the repetition that follows an RP, the normal part two of each operation is omitted, except that (PAK) is increased by one. The contents of PAK is initially the complement of the quantity jn that occupies the U-address section of the RP instruction, so (PAK) acts as an operations counter, containing the complement of j, (n - r) after r executions. In the particular case in which RP precedes a Threshold Jump or an Equality Jump, the complement of (PAK) is transmitted via X to the right-hand 15 places of Q if a jump occurs.

The Scale Factor instruction was designed to handle the case u = a, in which (A) can be a 72-bit number whose most significant bit may be either in A_R or in A_L. After the initial shift of 36 places a shift counter is set to 36 and then follows the sequence 36, 35, . . . 0, 71, 70, . . ., 37 as the subsequent shifts are made.

3.3 TRANSMISSIVE INSTRUCTIONS

In the following seven instructions u is an acquisition address and v is a transmission address. In other words all these instructions start by obtaining (u) in X and end by transmitting a word from X to v.

TPuv Transmit Positive: Replace (v) with (u).

TNuv Transmit Negative: Replace (v) with the complement of (u).

TMuv Transmit Magnitude: Replace (v) with the absolute magnitude of (u).

TUuv Transmit U-address: Replace the 15 bits of (v), designated by v_{15} through v_{29}, with the corresponding bits of (u), leaving the remaining 21 bits (v) undisturbed.

TVuv Transmit V-address: Replace the right-hand 15 bits of (v), designated by v_0 through v_{14}, with the corresponding bits of (u), leaving the remaining 21 bits of (v) undisturbed.

ATuv Add and Transmit: Add D(u) to (A). Then replace (v) with (A_R).

STuv Subtract and Transmit: Subtract D(u) from (A). Then replace (v) with (A_R).

Let us first consider the TP instruction in some detail. It involves two stages, acquisition of (u) in X and transmission from X to v. If u and v are both in ES, the number of clock pulses required is 24. However, u and v can be a or q, in which case the number of clock pulses is reduced. When a TP instruction is repeated, the time usually required for acquiring the instruction is saved. For example, the time required for such a repeat applied from ES to ES is 15 clock pulses.

A repeated TP may be used to effect a block transfer from MD to ES. In fact the entire organization of a block transfer is accomplished by the two instructions.

RP3nw

TPuv

The effect of these two instructions is to transfer the block of words at addresses u + i in MD to addresses v + i in ES, where i = 0, 1, 2, ..., (n-1), and then to cause the program to jump to address w. The speed of the transfer will depend on the interlace that is in use, but since each execution of the repeated TP requires, at maximum, only 16 clock pulses, or 32 microseconds, it would be possible to pick up successive words from positions in MD that are four cells, or 32 microseconds, apart. This means that with four-interlace the block transfer can be achieved with an interval of only 32 microseconds between successive words. Similar considerations apply to block transfers from ES to MD.

It should be noted that the execution of TNuv does not involve A unless either u = a or v = a. This is the main advantage gained by using the one's complement number system, which allows the change of sign to be achieved by simply complementing (X), whereas in a two's complement system the change of sign would have to be achieved by subtraction from zero. The advantage is twofold; it allows the negative of (u) to be transmitted to v without changing (A), and allows the transmission to be executed in a shorter time than would be possible if A were involved in the change of sign. By repeating a TN order, block transfers with a change of sign can be achieved at the same high speed as the positive block transfers using TP. The same remarks apply to TMuv, because the conditional change of sign is made in X.

The instructions TU and TV are, of course, intended for the modifica-

tion of execution addresses of stored instructions. It has already been remarked that the v-address of these instructions may not be q or a.

The add and subtract instructions AT and ST need no special comment. Their behavior is straightforward in all cases in which q and a are used as execution addresses.

3.4 Q-CONTROLLED INSTRUCTIONS

The following three instructions are transmissive in that they start by obtaining (u) and end by transmitting to v. Their common characteristic feature is the fact that (Q) is used as a mask to control the operations.

QTuv — Q-controlled Transmit: Form in A the number L (Q) (u). Then replace (v) by (A_R).

QAuv — Q-controlled Add: Add to (A) the number L(Q) (u). Then replace (v) by (A_R).

QSuv — Q-controlled Substitute: Form in A the quantity L(Q)(u) plus L(Q')(v). Then replace (v) with (A_R). The effect is to replace selected bits of (v) with the corresponding bits of (u) in those places corresponding to 1's in Q. The final (v) is the same as the final (A_R).

The following peculiarities arise from the use of q and a as execu-execution addresses:

QSua	L(Q)(u)	is left in A.
QSqa	S(Q)	is left in A.
QSaa	$L(Q)(A_R)$	is left in A.
QSqq	-0 (all ones)	is left in Q.

In QSua it is assumed that the address u is in ES or MD.

3.5 REPLACE INSTRUCTIONS

The common characteristic of the following five instructions is that the execution address u is used both as an acquisition address and also as a transmission address. Each of the instructions replaces (u) by a word that is derived from (u). In all cases the operation starts with the acquisition of u; the first three instructions involve a second execution address, while the remaining two involve shifts.

RAuv — Replace Add: Form in A the sum of D(u) and D(v). Then replace (u) with (A_R).

RSuv — Replace Subtract: Form in A the difference D(u) - D(v). Then replace (u) with A_R.

CCuv — Controlled Complement: Replace (A_R) with (u) leaving (A_L) undisturbed. Then complement those bits of A_R that correspond to ones in (v). Then replace (u) with (A_R).

LAuk — Left shift in A: Replace (A) with D(u). Then left circular shift (A) by k places. Then replace (u) with (A_R). If u = a, the first step is omitted, so that the initial content of A is shifted.

LQuk — Left Shift in Q: Replace (Q) with (u). Then left circular shift (Q) by k places. Then replace (u) with (Q).

The RA and RS instructions are useful in cyclic programs, when the execution addresses of certain instructions have to be modified by prescribed amounts. They also provide another example of the efficiency of the Repeat (RP) instruction. With the notation used in 3.2 above the two instructions

RP3nw

RAuv (or RSuv)

will put the n sums $a_i + b_i$ (or the n differences $a_i - b_i$) in the memory boxes

u_i previously occupied by a_i. This amounts to the addition (or subtraction) of two 1 x n matrixes, or of two n x 1 matrixes.

Apart from its effect on A, the Controlled Complement instruction (CC) may be thought of as equivalent to adding (u) and (v) without carries, and putting the result in (u).

The only peculiarities arising from using q and a as execution addresses are as follows, the address u being in ES or MD:

RAua — Twice the number D(u) is formed in A. Then (u) is replaced by (A_R).

RAqa — Twice the number D(Q) is formed in A. Then (u) is replaced by (A_R).

RAaa — Twice the number $(A_R)_i$ is formed in A.

RSua — Both (A) and (u) are replaced by zero.

RSqu — Both (A) and (Q) are replaced by zero.

RSqq — Both (A) and (Q) are replaced by zero.

CCua — (A_L) is unaltered. (A_R) and (u) are replaced by zero.

CCqa — (A_L) is unaltered. (A_R) and (Q) are replaced by zero.

CCaa — (A_L) is unaltered. (A_R) is replaced by zero.

Incidentally, as one would expect, the instruction RSaa clears all 72 bits of the accumulator.

3.6 SPLIT INSTRUCTIONS

The following four instructions make it possible to handle double-precision words in A. They all start by acquiring a word from u, and then perform a left circular shift in A.

SPuk Split Positive entry: Form S(u) in A. Then left circular shift (A) by k places.

SNuk Split Negative entry: Form in A the complement of S(u). Then left circular shift (A) by k places.

SAuk Split Add: Add S(u) to (A). Then left circular shift (A) by k places.

SSuk Split Subtract: Subtract S(u) from (A). Then left circular shift (A) by k places.

If a programmer wishes to clear A_R without changing (A_L), or to clear A_L without changing (A_R) he may do so with the instructions SSao or SPao of 3.6.

3.7 TWO-WAY CONDITIONAL JUMP INSTRUCTIONS

In the following three instructions, u and v are two alternative addresses from which the next instruction, NI, may be obtained.

SJuv Sign Jump: If $A_{71} = 1$, take (u) as NI.
If $A_{71} = 0$, take (v) as NI.

ZJuv Zero Jump: If (A) is not zero, take (u) as NI.
If (A) is zero, take (v) as NI. In either case, leave (A) in its initial state.

QJuv Q-Jump: If $Q_{35} = 1$, take (u) as NI. If $Q_{35} = 0$, take (v) as NI. Then, in either case, left circular shift (Q) by one place.

The distinction between addresses u and v in the above instructions is easily remembered because in each case a program jump to v is called for by something, either A_{71} or (A) or Q_{35}, being zero.

The Q-Jump is an unusual feature of the ERA 1103 computer. It allows a programmer to make the control of a program sequence depend on the succession of ones and zeros in a single word. This fact enables one to apply the Q-Jump in problems involving control by a random-sequence of bits.

3.8 ONE-WAY CONDITIONAL JUMP INSTRUCTIONS

In the following three instructions u is an acquisition address and v is an address from which the next instruction NI will be obtained, if some condition is satisfied.

IJuv Index Jump: Form in A the difference D(u) minus 1. Then, if $A_{71} = 1$, continue the present sequence of instructions; if $A_{71} = 0$ replace (u) with (A_R) and take (v) a s NI. If u = a, the initial acquisition of (A_R) is suppressed so that (A) -1 is formed in A, rather than $D(A_R) - 1$.

TJuv Threshold Jump: If D(u) is greater than (A) take (v) as NI; if not, continue the present sequence. In either case, restore (A) to its initial state.

EJuv Equality Jump: If D(u) equals (A) take (v) as NI; if not, continue the present sequence. In either case restore (A) to its initial state.

In the description of the RP instruction special reference was made to the use of RP immediately before TJ or EJ. Consider the effect of the two instructions.

RP2nw

TJuv

and suppose that quantities a_i are stored in addresses u + i, where i = 0, 1, 2, ..., n. If all the quantities a_i are less than the number (A), and TJ instruction will merely be repeated n times with successive reference to the addresses u + i, and the program will then jump to address w. On the other hand as soon as a quantity a_i is found that is greater than (A) the program jump to address v is made, and the value of i that gave rise to this jump is indicated by the number (n - r) that is put in right-hand four octal digits

of Q. Here r is number of times TJ has been executed in the present repeat sequence. A similar remark can be applied to EJ wherein one is testing for the equality of D(u) and (A).

The index jump, IJ, further illustrates the versatility of the 1103 Computer instructions. In performing an iterative cycle it is often necessary to count the number of times through the cycle and, when a preassigned count is reached, make an exit from the cycle. The index jump accomplishes this by means of a single instruction.

3.9 ONE-WAY UNCONDITIONAL JUMP INSTRUCTIONS

The following two jump instructions are termed "unconditional" because the execution of the program jump to address v does not depend on applying a test to the transient state of affairs in A. It should be noted, however, that the use of the U-address in MJ makes this instruction equivalent to four separate unconditional jump instructions, three of which can be operative or made inoperative at will by the use of manual switches. Thus, the term "unconditional" is not strictly correct, in that the setting of a switch is a form of condition.

MJjv — Manually selective Jump: If the number j is zero, take (v) as NI. If j is 1, 2 or 3, and the correspondingly numbered MJ selecting switch is set to "jump", take (v) as NI; if this switch is not set to "jump", continue the present sequence.

RJuv — Return Jump: Suppose that the instruction RJuv was acquired from address x. Then (1) (PAK) = x + 1 is copied in the right-hand fifteen places at address u, without affecting the remaining bits at that address. (2) (PAK) is then replaced by v, so that (v) will be taken as NI.

The principal purpose of the RJ instruction is to enable the programmer both to cause the program to jump to a subroutine and also to arrange for the program to jump back after completing the subroutine to the sequence that was interrupted by the first jump. Suppose for example that a subroutine is prepared that will obtain the square root of the 72-bit number in A and will leave the result in A. Suppose this subroutine starts at address s and ends at address e with an unconditional jump MJ (j = 0). Then the single instruction

RJes may be used exactly as if it were a special instruction for the purpose of obtaining the square root of (A) and leaving the result in A.

The address a may not be used as the U-address of an RJ instruction.

3.10 INSTRUCTIONS FOR MAGNETIC TAPE STORAGE, MT

In the MT instructions the quantity j designates one of four MT units. The quantity n denotes the number of 32-word blocks to be traversed.

AMjn Advance Magnetic Tape: Move the magnetic tape of MT unit j in the forward direction by n blocks.

BMjn Back Magnetic Tape: Move the magnetic tape of MT unit j in the backward direction by n blocks.

RMjnv Read Magnetic Tape: Read n blocks from MT unit j (running forward) to 32 n consecutive addresses in ES starting with v.

WMjnv Write Magnetic Tape: From 32 n consecutive addresses in ES, starting with v, write n blocks on MT unit j (running forward).

In the execution of AM and BM, the computer program may proceed as soon as the tape movement has been initiated. Should another instruction referring to the same tape unit occur before the tape movement is completed, the program would then automatically be caused to wait.

It should be noted that further magnetic tape storage could be operated as external equipment, using the instructions EF, ER, EW that will be described below. When operating a magnetic tape unit in this fashion, a programmer would have to provide computer instructions to govern the handling of individual tape characters, but he would be able to make the computer do other things in the intervals between the acquisition or transmission of characters. In the case of the MT units that are controlled by the orders AM, BM, RM, WM, the computer is fully occupied during the execution of the block transfers called for by RM and WM, and can do nothing else while they are in process.

3.11 STOPPING AND STARTING

Instructions that cause the computer to stop are the following:

MSjv Manually selective Stop: If j = 0, stop the computer operation and provide suitable indication. If j = 1, 2, or 3 and the correspondingly numbered MS selecting switch is set to "stop", stop the computer operation and provide suitable indication. Whether or not a stop occurs, prepare to take (v) as the next instruction.

FS Final Stop: Stop computer operation and provide suitable indication.

In addition to programmed stops, the computer operation may be stopped for any one of the following reasons, a suitable indication being given in each case.

1. Imminent overflow of the full 72-bit capacity of A (tested prior to a Multiply Add instruction).
2. Attempt to obtain a quotient that would overflow the 36-bit capacity of Q.
3. Use of the address q (of Q) as the V-address in TU, TV, SF, or RJ.
4. Occurrence of an improper comand code that does not correspond to any of the computer instructions.
5. Use of the address a (of A) for an instruction.

The computer may be started in the "operate" mode in either of two ways, manually selected.

Start from MD: If this option is selected, the computer takes its first instruction from F_0, a fixed address (40000) in MD. In order to start in this manner, a

loading operation must previously have taken place, and, in particular, an appropriate instruction inserted in F_0.

Start from MT: If this option is selected, the computer will first automatically execute the instruction Read Magnetic tape with j = 0, n = 0001, and v = 00000, and then set PAK to take (00000) as the next instruction. In this way, the first block on an MT unit may be used to direct the loading of information from magnetic tape, without any preliminary loading.

3.12 EXTERNAL EQUIPMENT - INPUT AND OUTPUT

The basic instructions for handling input and output and for the control of external equipment are as follows:

EF-v External Functions: Select a unit of external equipment and perform the function indicated by (v)

ERjv External Read: If j = 0, replace the right hand 8 bits of (v) with (IOA); if j = 1, replace (v) with (IOB). If a step-by-step unit is involved, advance it one step.

EWjv External Write: If j = 0, replace (IOA) with the right hand 8 bits of (v); if j = 1, replace (IOB) with (v). Cause the previously selected unit to respond to the information in IOA and IOB.

It should be noted that IOA has only 8 bits. These are associated with the right-hand 8 bits of (v) in ER and EW.

The interpretation of the address v of an EF instruction can be arranged to suit particular applications. The variety of possible applications is very considerable, and will not be discussed fully here. For example EF-v_1 might select a rotating shaft and call for a measurement of the angular position to be obtained and put in IOB, whence it could subsequently be read into the computer by ERj_1v. In such an application once the instruction EF has been given the computer can go on computing while the measurement is being made. Another instruction EF-v_2 might select the same shaft and cause it to be rotated. This is a crude example of the means by which the 1103 computer can be made to act as the central control of a physical system.

Another obvious application of an EF instruction would be to select a cathode ray tube on which a visual presentation is to be made; two EW instructions could be used to set the horizontal and vertical deflections and to illuminate a spot, so that a programmer could arrange to produce any desired visual display; another EF instruction might control a camera to obtain a photograph of the display.

An important application is the acceptance of a stream of information that is neither controlled by the computer in a step-by-step fashion, nor synchronized with the computer. Suppose for example that a particular EF instruction selects a photo-electric reader and causes it to read successive characters from punched paper tape and to insert them in IOA at its own speed until the tape motion is stopped by another EF instruction. Provided that the programmer knows the maximum speed of the tape, he can insert enough ER instructions in his program to make sure that no tape character is missed. If an ER instruction arrives before the corresponding character has been transferred from tape to IOA, the computer waits, this being ensured by an interlock. Essentially what happens is that, whenever a new character is transferred from tape to IOA, and "indicating flip-flop" (IFF), is set to indicate the presence of new information. When an ER instruction, that refers to IOA, finds IFF not set, the computer will wait for IFF to be set before executing the ER. Further the execution of an ER causes IFF to be reset to indicate the absence of new information. The next character from tape should, therefore, find IFF in the reset state; if IFF still indicates the presence of new information, the tape has got ahead of the programmer, and an alarm signal must be given.

The above example shows that the 1103 is capable of handling many forms of input. It is possible to put information into the computer as binary numbers, which is convenient in many cases of on-line operation with measurements

of physical quantities, or in any coded form that may be convenient for a particular application. The flexible logic of the 1103, and in particular the instructions QT, QA, QS, CC, make it easy for a programmer to arrange for any necessary conversions and to get them done fast, often while input is in progress. Many possibilities spring to mind: Photoelectric reading from punched cards to IOB in standard card code; direct input of information from a teletype circuit, or from some other communications link; direct-on-line input of measurements of variable parameters in an experimented system such as a wind tunnel; magnetic tape or magnetic wire units; and so on.

Similarly there are a great variety of possible output arrangements. Broadly speaking the 1103 system of handling external equipment makes it possible for the computer to work with any external equipment that can be activated by signals in binary code and that can either accept information from a flip-flop register or deposit information in a flip-flop register. Further it is felt that in cases in which a large scale computer is not permanently allocated to work of a special type, it will often be far more efficient to use the facilities of the 1103 for code conversion, rather than build special external equipment for each type of conversion.

In addition to the basic input-output instructions EF, ER, EW, the 1103 has special instructions for punched paper tape output and for typewriter output. There are also special facilities for input from punched paper tape.

PR-v PRint Replace (TWR) with the right-hand bits of (v). Cause the typewriter to print the character corresponding to the 6-bit code.

PUjv PUnch Replace (HPR) with the right-hand 6 bits of (v). Cause the punch to respond to (HPR). If j = 0, omit seventh level hole; if j = 1 include seventh level hole.

In connection with the PRINT instruction, the typewriter characters represented by octal code 77 will be designated as "illegal code" indicators. By means of a manual selecting switch, the occurrence of an illegal typewriter code will cause the computer (1) for PR-v to stop and indicate a fault condition, or (2) for PR-v to print the "illegal code" indicator and continue operation. In the case of PUjv, the high-speed punch will accept and punch all possibilities, without restriction.

The special facilities for input from punched paper tape utilize a photoelectric tape reader. Seventh level coding is used to control the assembly of six consecutive tape characters to form a 36-bit word and the transmission of consecutive words to consecutive addresses in storage starting with a prescribed address that is punched on the tape. This feature allows programs of any length to be placed in various parts of the magnetic drum or electrostatic storage without making use of a pre-stored program to control the operation.

4.0 1103 INSTRUCTION EXECUTION TIMES (FOR ES STORAGE ONLY)

All entries represent the number of clock periods required to perform the indicated instruction. (For Normal mode from MP6 to MP6; for Repeat, from MP7 to MP7.) A clock period is approximately 2.0 microseconds.

INSTRUCTION	OCTAL CODE	CONDITION	NORMAL		REPEATED	
			MIN.	MAX.	FIRST	SECOND ETC.
TPuv	11	-	22	25	17	16
TMuv	12					
TNuv	13					
TUuv	15					
TVuv	16					
EF-v	17	-	16	19	11	12
RAuv	21	-	38	41	33	32
RSuv	23					
CCuv	27		30	33	25	24
SPuk	31	$k \geq 2$	18+k	21+k	13+k	$11+k+p_1$
SAuk	32					
SNuk	33	$k = 0,1$	18+k	21+k	15	16
SSuk	34					
ATuv	35	-	26	29	21	20
STuv	36					
RJuv	37	-	22	25	-	-
IJuv	41	Jump	30	33	25	24
		No Jump	24	27	19	20
TJuv	42	-	23	26	-	-
		No J. Term.	-	-	18	16
		J. Term.	-	-	23	21
EJuv	43	-	29	32	-	-
		No J. Term.	-	-	24	24
		J. Term.	-	-	29	29
QJuv	44	-	10	13	-	-
MJuv	45					
SJuv	46					
ZJuv	47	-	16	19	-	-

4.0 1103 INSTRUCTION EXECUTION TIMES (continued)

INSTRUCTION	OCTAL CODE	CONDITION	NORMAL		REPEATED	
			MIN.	MAX.	FIRST	SECOND ETC.
QTuv	51	-	26	29	21	20
QAuv	52	-	26	29	21	20
QSuv	53	-	46	49	41	40
LAuk	54	-	$24+k+p_2$	$27+k+p_2$	$19+k+p_2$	$18+k+p_2$
LQuk	55	-	$23+k+p_3$	$26+k+p_3$	$18+k+p_3$	$17+k+p_3$
MSjv	56	-	10	13	-	-
FS--	57	-	9	12	-	-
PR-v	61	-	17	20	12	12
PUjv	63	-	17	20	12	12
RMjnv	64	-	-	-	-	-
WMjnv	65	-	-	-	-	-
AMjn-	66	-	14	17	-	-
BMjn-	67	-	14	17	-	-
MPuv	71	-	$60+a$	$63+a$	-	-
MAuv	72	-	$96+a$	$99+a$	$91+a$	$89+a+p_4$
DVuv	73	-	$242+4a_{71}$	$249+4a_{71}$	-	-
		Min.	-	-	$237+4a_{71}$	$236+4a_{71}$
		Max.	-	-	$241+4a_{71}$	$240+4a_{71}$
SFuv	74	-	$63+\beta+p_5$	$66+\beta+p_5$	$58+\beta+p_5$	$57+\beta+p_5$
RPjnw	75	-	33	36	-	-
ERjnv	76	-	18	21	9	12
EWjnv	77	-	16	19	11	12

$p_1 = [1-k] \bmod 4$

$p_2 = [2-k] \bmod 4$

$p_3 = [3-k] \bmod 4$

$p_4 = [q_0+q_{35}+3] \bmod 4$

$p_5 = [3+k] \bmod 4$

$a = 11q_{35}+3q_0+4(q_1+q_2+\ldots+q_{34})$

$\beta = [36-k] \bmod 72$