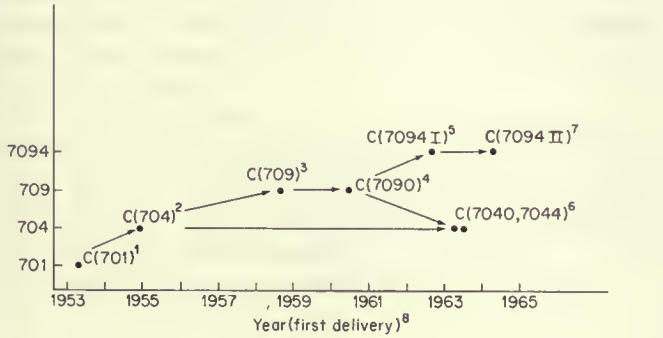
Section 1

The IBM 701-7094 II sequence, a family by evolution

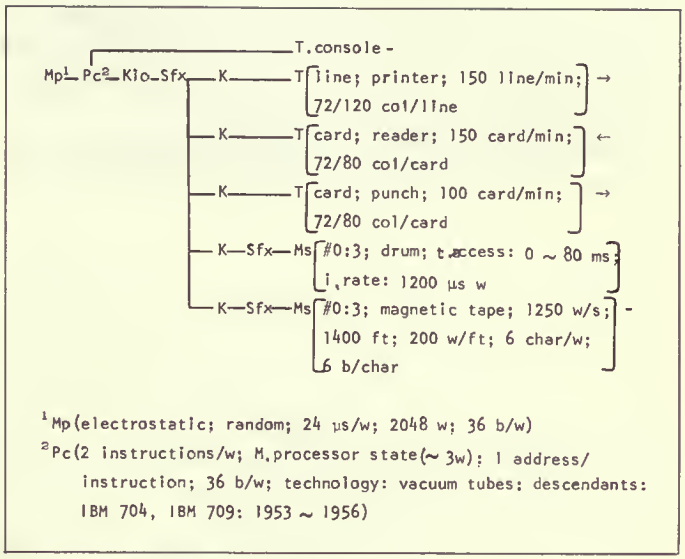
The IBM 701, 704, 709, 7090, 7040, 7044, 7094 I, and 7094 II sequence relationship is shown in Fig. 1. The group is not a compatible series. The IBM 701 [Astrahan and Rochester, 1952; Buchholz, 1953] is a forerunner of the series; all except the 701 are painfully compatible. The sequence is included because the 7090 is a reference or benchmark of scientific-computer power. All machines use 36-bit words. The 701 stores two instructions/word in the same manner as the IAS computer (Chap. 4), whereas all others in the sequence store only one instruction/word. The 701, 704, and 709 are first-generation, vacuum-tube technology; the rest are second-generation.

The IBM 7094 II description given in Chap. 41 is based directly on information in the Programming Reference Manual, but the Appendices of that chapter give the ISP of the Pc, a Pio, and a K as inferred by the authors of this book. The description of the Pc gives the instructions in the 704 and 7044



[1]C(IBM 701; vacuum tube; 36 b/w; 2 instruction/w; *similar to IAS/von Neumann C; progrom controlled I/O doto transmission*; Mp (electrostatic; 30 μs))
[2]C(IBM 704; vacuum tube; 36 b/w; 1 instruction/w; *program controlled I/O; traps, interrupts*; Mps (AC, MQ, 3 Index Registers Instruction Counter) Mp (core; 12 μs))
[3]C(IBM 709; 704 *upward compotible; Pio controlled doto transmission*)
[4]C(IBM 7090; transistor; Mp (core; 2.18 μs))
[5]C(IBM 7094 I; *upword* 7090 *compotible; overlopped memory*; Mps (7 Index Registers; Pio ('7909); Mp (2 μs))
[6]C(IBM 7040, 7044; *Pio, progrom controlled I/O*; Mp(8,2 μs))
[7]C(IBM 7094 II; Mp (1.4 μs))
[8]Adam's Associotes Computer Chorocteristics Quorterly

Fig. 1. Relationships among IBM 701, 704, 709, 7094 series.



[1]Mp(electrostatic; random; 24 μs/w; 2048 w; 36 b/w)
[2]Pc(2 instructions/w; M.processor state(~ 3w); 1 address/instruction; 36 b/w; technology: vacuum tubes; descendants: IBM 704, IBM 709: 1953 ~ 1956)

Fig. 2. IBM 701 PMS diagram.

to show an evolution. However, the major evolutionary change does not appear in Pc's ISP but in the PMS structure.

The 704 structure, like that of the 701 (Fig. 2), provides only for peripheral transfers to primary memory via Pc under programmed control with no interrupt system. As such, only one T or Ms could operate easily at a time. The 709 introduced the Pio('Data Channels) to improve the ability to transfer data between Mp and Ms without requiring Pc intervention. Concurrent operation of several I/O devices is carried out by multiple Pio's along the lines of the 7094 II PMS structure (Fig. 1, Chap. 41, page 518). However, the utilization of the data channels tends to be rather low, particularly when the data channel is controlling very slow devices (e.g., card equipment and line printers). When operating a high-speed tape unit at 90,000 × 6 bits/sec the utilization of the data channel is still only approximately 3 percent. A program interrupt method of data transfers would have been sufficient.

The incompatibility among the machines, especially the 7090-7040-7094, is disheartening, both from the point of view of a user and an engineer. The incremental hardware needed

to achieve compatibility is inexpensive when the system price is considered. Also, the incremental changes in the ISP do little to increase the Pc performance. Compared with the 704, the extensive order code of the 7094 shows an evolution in which for marketing, emotional, or analytic reasons new instructions were added. The index registers and their instructions are a good example of this trend. The 7094 has a very general set of index-register transmission instructions; if implemented properly, they are probably easier to provide than the original 704 instructions.

In the implementation of the double-precision floating-point hardware, the sense-indicator register is needed for temporary storage. Thus a user has to preserve this register when double-precision floating-point instructions are given. The reason for this undoubtedly relates to field modifications and cost. In an original design this would be inexcusable; in this case double-precision floating point is undoubtedly worth the loss of sense indicators.

All in all, the designers of the 704-7094 II provided increased generality through evolution. They gradually ran out of patching time, technology, instruction encoding space, and memory addressing bits, while exceeding compatibility constraints. It was indeed time to create the IBM System/360.

Chapter 41

The IBM 7094 I, II

Introduction

The IBM 7094 I and 7094 II computers are the last of a series of computers beginning with the IBM 704 (Fig. 1, page 515). The series is an outgrowth of the IBM 701. Although the series is designed for scientific (arithmetic) calculations, its speed and structure allow it to be used for general-purpose computation. Business-type processing which uses string data is efficiently handled by conversion into fixed-length fields at input and output. From about 1956 to 1966 the family was the standard of large computers in the United States, there being approximately 20 701, 50 704, 20 709, 50 7090, 130 7094 I, 125 7094 II, 120 7040, and 120 7044 computers in existence.

The PMS structure is a single central processor (Pc) with multiple input/output processors (Pio's) (for all except the 701 and 704). The Pio's provide for multiple transfers to primary memory (Mp) at high information flow rates. The structure allows for duplex connection to terminal (T) or secondary-memory (Ms) control (K). This provision permits the system to be used in real-time applications requiring significant computation, high-data-rate transfers with other systems, and high availability. However, the system was not initially designed for time sharing and multiprogramming use, and the attempt to so use it required modification [Corbato et al., 1962].

The word length is 36 bits. There is one single-address instruction/word. In all but the 7094 the processor interprets instructions serially. In the 7094 one register instruction look-ahead is used. The Pc has index registers, the 704 being the first IBM computer to use them. Their number increased from three in the 704 ~ 7090 to seven in the 7094, as their usefulness became apparent.

Structure

A simple tree-structured IBM 7094 I using PMS is shown in Fig. 1 and using a conventional block diagram in Fig. 2.

Primary memory (Mp) and P-Mp switch

The primary memory, Mp('7302 Core Storage), has a capacity of 32,768 36-bit words with a cycle time of 2 microseconds. The actual memory has a 72 + 1 parity bit word for even and odd addresses of 36-bit words. A request for two 36-bit words can be acknowledged in one 2-microsecond memory cycle. Thus Mp is Mp('7302 Core Storage; 2 μs/w; 16384 w; (72, 1 parity) b/w) for the 7094 I, and Mp(1.4 μs/w; 16384 w; (72, 1 parity) b/w) for the 7094 II.

The S('7606 Multiplexor; time multiplexed) provides access to Mp from any one of nine P's. Only Pc can request two 36-bit words at a time from Mp for instruction look-ahead and double-word operations. There can be only one Pc in the system.

Processors, P

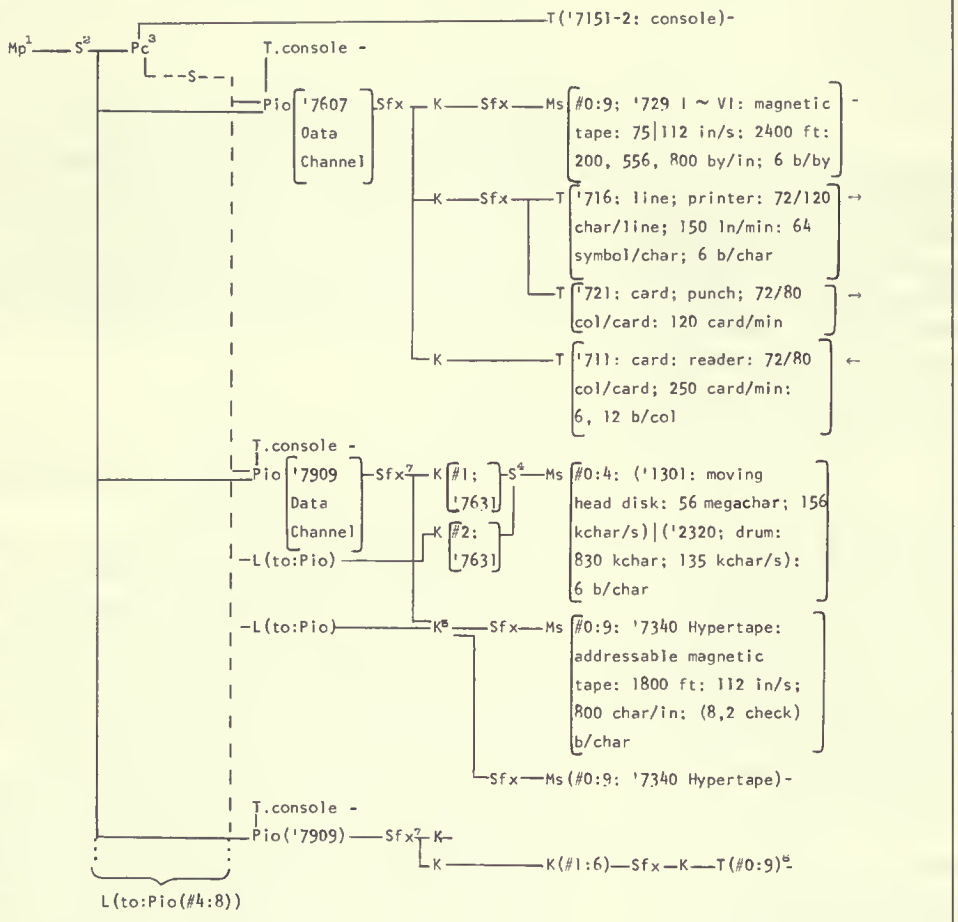
Three processors are described: Pc('7109, 7110 Central Processing Unit/CPU), Pio('7607 Data Channel), and Pio('7909 Data Channel).

All P's behave similarly in that Pc instructions and Pio commands[1] are fetched (or requested) from Mp and then interpreted in P. An instruction location counter in P addresses the next instruction. A processor instruction may, in turn, require the processor to access Mp for data, to perform transfers, to modify its state, etc. Although structurally the P's are similar, organizationally the Pc is superior to the Pio('Data Channel)'s; Pc issues programs to Pio's and start and stops (controls) Pio's.

Two-way communication is required between Pc and the Pio's. Tasks (jobs or programs) for Pio's are first set up in Mp by Pc. Pc then demands that Pio execute the program independently under its own control. Initialization takes place when Pc sets the instruction counter of a Pio. Upon task completion in Pio, an interrupt request is sent to Pc from Pio.

Below we first give a description of the Pc. Then the Pio('7909) is presented in detail and the Pio('7607) is outlined. The reader should compare the two Pio's. The Pio('7909) is a later design than the Pio('7607). It interprets instructions for the block of data being transferred and issues instructions to the KMs or KT. The earlier Pio('7607) interprets the instructions for controlling the information being transferred; the Pc interprets and issues the instructions to KMs or KT. The 7909 is therefore able to control more closely a T or Ms using a single program without need for Pc intervention.

[1]IBM attempts to distinguish between Pc and Pio's terminologically by "instruction" and "command." We make no such distinction in the following discussion; P's interpret instructions.



[1] Mp(core; 16384 w; (72, 1 parity) b/w; 2.0 μs/w)

[2] S(time multiplexed; '7606 Multiplexor; 1 M: 9 P: radial; location: central)

[3] Pc('7109,7110 Central Processing Unit; 1 instruction/w; 1 address/instruction; Mps(12w); data: si,bv,sf,suf,df,duf,fr.i: technology: transistors: antecedants: IBM 704,709, 7090; descendants:IBM 7040,7044,7094 II; 1962 ~ 1966)

[4] S(fixed; from: 2 K; to: 5 Ms; concurrency:2)

[5] K(#1,2; '7640)

[6] T := (T(1011; paper tape; reader; 500 char/s; 6,7,8 b/char)|T('1014; paper tape; punch)|T(Teletype)|T(typewriter))

[7] S(fixed; 2 K)

Fig. 1. IBM 7094 I PMS diagram.

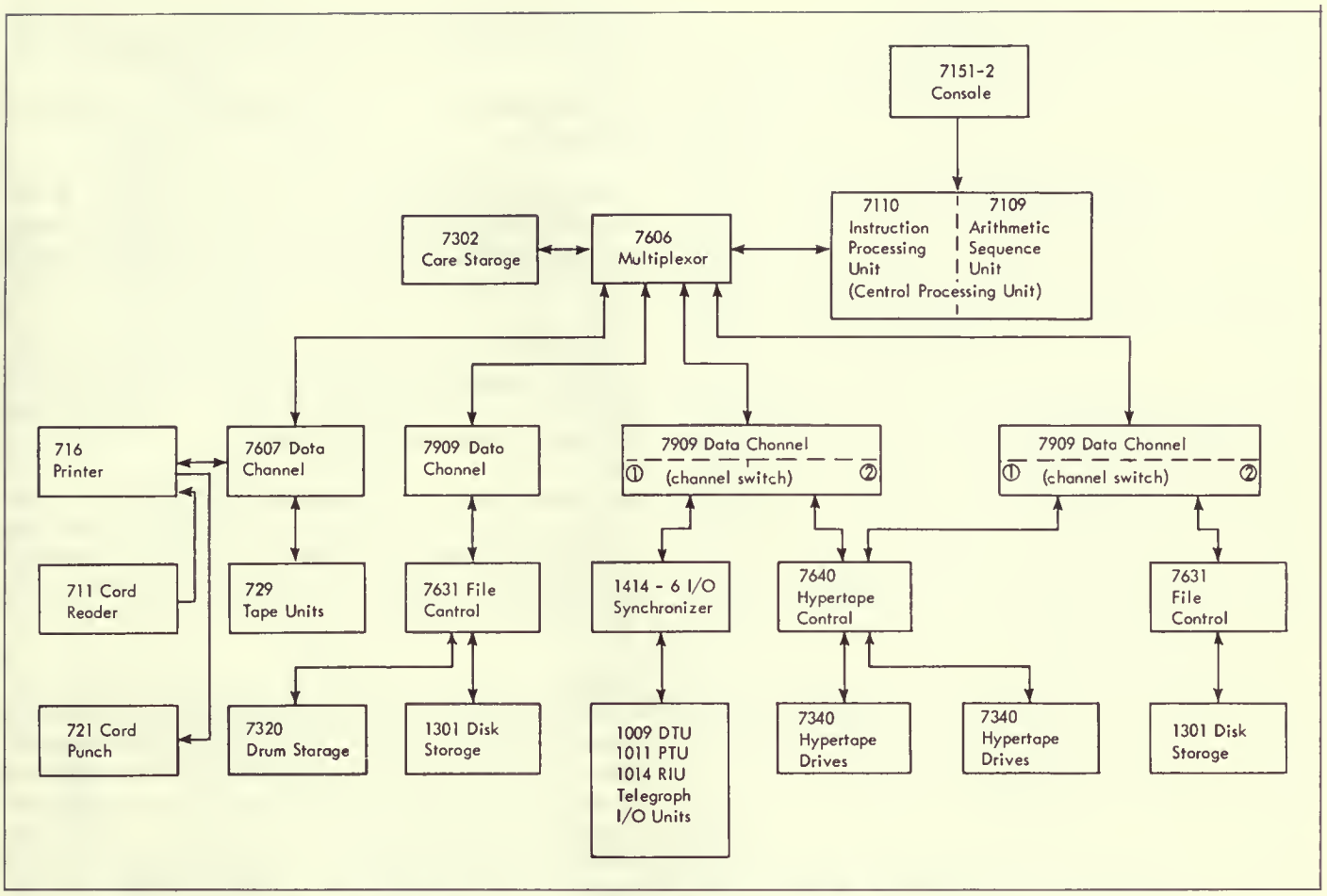


Fig. 2. IBM 7094 data-processing system configuration. (*Courtesy of International Business Machines Corporation.*)

Central processing unit, Pc

The Pc has three physical parts: the T('7151-2 Console), the D('7110 Instruction Processing Unit), and the K('7109 Arithmetic Sequence Unit). In terms of gross PMS parameters the 7094 I can be described similar to footnote three of Fig. 1 as

```
Pc['7109, 7110, 7151-2 Central Processing Unit/CPU;
   36 b/w; 1 address; 1 instruction/w;
   data: (si, bv, sf, suf, df, duf, fr.i);
   number representation: sign, magnitude;
   Mps('Accumulator, 'Multiplier_Quotient, 'Index_Registers
      [1:7], 'Sense_Indicators, 'Instruction_Counter, 'Trap_En-
      able⟨1:12⟩, 'miscellaneous_bits⟨1:7⟩)
```

The Pc will be discussed in two parts: the Register-transfer level implementation and the Instruction-set Processor. These are partially redundant, but they offer another opportunity to compare the two types of descriptions. The Pc hardware will be described by first giving the registers and the interregister transfer paths. Then the process by which instructions are interpreted will be described. (Interpretation occurs in a distinct set of memory cycles, called instruction/I, execute/E, logic/L, and buffer/B, which are sometimes mentioned in describing registers and will be fully discussed later.)

Processor registers and mode bits registers

Figure 3 gives the Pc registers and the data transfer paths. Both the ISP registers (denoted by °) and the temporary registers are given. The ISP registers and modes are controlled by the program.

Instruction counter (IC)°. The Instruction Counter, IC, is 15 bits. It is used by the processor to locate the next instruction in Mp. Once the program is started, the IC can be set to an address specified by a transfer instruction. For most instructions, the IC is stepped sequentially by I with each new instruction. The IC is normally advanced at the end of each instruction (I cycle).

Instruction backup register (IBR). The Instruction Backup Register, IBR, is a 36-bit register, ⟨S, 1:35⟩, and is used to buffer the next instruction. Pc attempts to have the next instruction available in IBR, since the Mp permits 72-bit transfers, thus avoiding an unnecessary reference to Mp. When the instruction reference is to an even location, the IBR is loaded with the contents of the next higher odd address after the contents of the even address have been placed in the Storage Register. The IBR is also used for fetching operands in double-precision operations.

Address register (AR). The Address Register, AR, is 15 bits and receives information from the Storage Register, Instruction Backup Register (at the beginning of a storage reference I or E cycle), Index Register, and Index Adder. The contents of the AR are sent to the Multiplexor Address Switch to select the core memory location.

Instruction register (IR). The 18-bit Instruction Register, IR, is divided into two parts: bits ⟨S, 1:9⟩ always contain the operation part of the instruction, and bits ⟨10:17⟩ form the Shift-counter Register. The Shift Counter is used during shifting, multiplication, division, and floating-point instructions. Bits ⟨10:17⟩ may also contain a sense instruction address, operation codes for those instructions which require an address part, and the class and unit codes for input/output instructions.

Storage register (SR). The 36-bit Storage Register, SR, stores information that comes from or goes to core storage.

Adders (not a register). The Adders furnish a 36-bit path for data going from the storage register to other registers in the processor.

Accumulator register (AC)°. The Accumulator Register, AC, is 38 bits (a 35-bit word with a 1-bit sign, and 2 bits for overflow conditions, P and Q). The AC is used to hold one factor during arithmetic or logical operations and to receive results from the adders.

Information may be shifted into the accumulator from the MQ, 1 bit at a time.

Multiplier-quotient register (MQ)°. The MQ Register is 36 bits. During a multiply instruction, MQ contains the multiplier; during a divide instruction, MQ receives the quotient. It can be shifted right or left, independently, or combined with AC into a 72-bit register.

Sense indicator register (SI)°. The Sense Indicator Register, SI, is 36 bits. SI is normally used as a set of binary program switches which can be set and tested. However, it is also used as a temporary register in double-precision arithmetic operations.

Index registers (XR)°. Seven 15-bit Index Registers, XRs, in the 7094 system are used for address modification. They are specified by the tag bits of an instruction (bits ⟨18:20⟩) and modify an address by adding the two's complement of their contents to the address. In the earlier 7090 (and 7044) only XR[1, 2, 4] are available.

Multiple tag mode°. In Multiple Tag Mode only Index Registers 1, 2, and 4 can be specified. The indexing function specified is determined by the "logical-or" of each index register specified. When not in Multiple Tag Mode, each 3-bit number selects one of seven index registers. The 1-bit Multiple-Tag-Mode Register maintains the state of the mode. The requirement for the two modes comes entirely from the need to maintain compatibility between the 704, 709, 7090, 7040, and 7044 (which have three index registers addressed as in Multiple Tag Mode) and the 7094 I and 7094 II which have seven index registers.

Tag register (TR). This temporary register holds the tag field of the instruction being executed and is used to select the Index Register being addressed.

Index adders (XAD) (not a register). A separate 15-position Index Adder is used for the Index-register operations. All storing, loading, changing, and modifying of Index Registers is via the Index Adders.

Accumulator overflow°. The Accumulator Overflow Indicator is turned on whenever a 1 passes into or through position P from position 1 of the AC as a result of the execution of a fixed-point arithmetic or a shifting instruction.

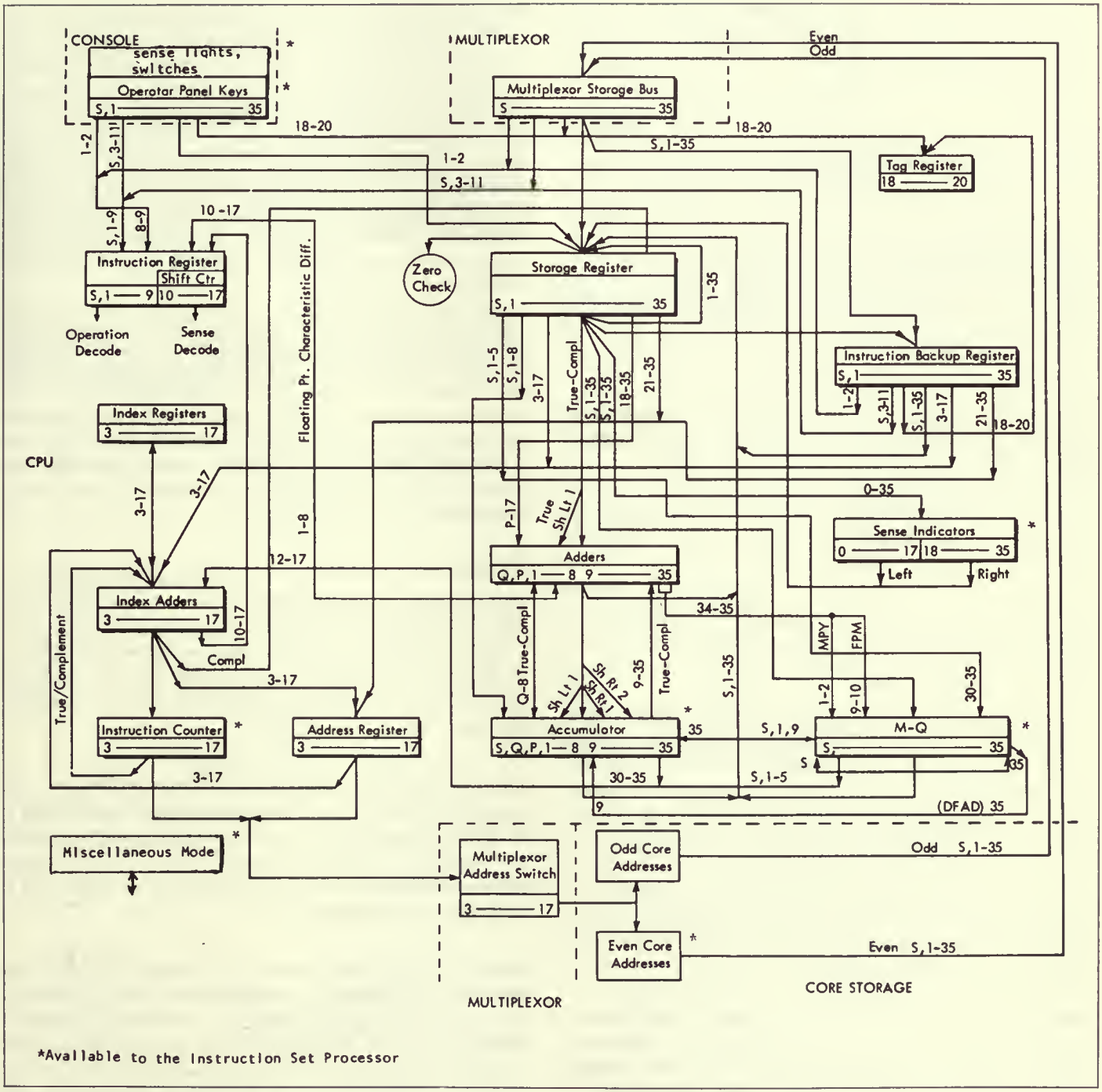


Fig. 3. IBM 7094 central-processing-unit information flow. (*Courtesy of International Business Machines Corporation.*)

Divide-check°. The Divide-Check Indicator is turned on, in fixed-point or floating-point division, if the magnitude of the number in the AC (dividend) is greater than or equal to the magnitude of the number in memory (divisor).

Input-output check°. The Input-Output Check Indicator (I-O check) is turned on by the attempted execution of an input/output instruction without first selecting an input/output unit.

Transfer trap mode°. The computer can be operated in a special Transfer Trap Mode. Operation in the Trap Mode permits the program to run at normal speed with interruptions of normal operation only at transfer points. At such points the location of the last sequential instruction is saved, and a transfer of control is made to a fixed location.

Sense switches°. Six Sense Switches are located on the console. They may be turned on or off manually, and there are instructions which sense them.

Sense lights°. Four Sense Lights are also on the console. Any one of these lights may be turned on, off, or the status tested by instructions.

Panel in-out switches°. These 36 switches on the console may be read by an instruction.

Instruction-set interpretation

The basic computer clock cycle is 2.0 μs in 7094 I and 1.4 μs in 7094 II, as dictated by Mp. Within the single 2- (or 1.4-) microsecond cycle, up to 10 sequential register transfers and/or data operations can take place, each of which transfers information among the Pc's registers; several operations may occur simultaneously. In Pc four different cycles are used: instruction/I, execute/E, logic/L, and buffer/B. The cyclic sequence of an instruction is fixed, always beginning with an I cycle and progressing to E, L, or B cycles, depending on the instruction. The number of cycles required for an instruction may vary from I (e.g., transfer) to 19 (e.g., double-precision floating-point divide).

Instruction cycle (*I*). The I cycle begins when IC furnishes the instruction location to Mp, via S('Multiplexor). The addressed instruction word taken from Mp goes to the Multiplexor Storage Bus (Fig. 3). From the Multiplexor Storage Bus the instruction is read into the Storage Register where it is separated into the operation portion and the address portion of the instruction word. The operation portion of the Storage Register goes into the Instruction Register, where the operation code is decoded and the execute control circuitry is set up to perform the operation specified by the instruction. The address portion of the instruction word, now located in the Storage Register, may be used directly. Normally, however, it goes to the Address Register and then to the Multiplexor Address Switch to locate the appropriate data word in Mp. If the address is to be modified, it is routed from the Storage Register to the Index Adders for Index-register modification. The modified address is then brought to the Address Register and on to the Multiplexor Address Switch to locate the data word in core storage.

Concurrently, during the same instruction cycle, a second instruction, located at the immediately higher odd-numbered Mp address location, is brought to the Instruction Backup Register/IBR. While in the IBR, the odd-numbered instruction is partially decoded to determine if it meets certain criteria for concurrent execution, thus saving a second Mp reference. If the instruction in the IBR cannot be executed with the current instruction, it is ignored in the current I cycle and is brought into the Storage Register on the next I cycle.

Execution cycle (*E*). The execution (E) cycle is used when a reference to core storage is needed. All instructions requiring an operand have an E cycle following the I cycle.

Indirect addressing of an instruction requires an extra E cycle. In other words, an instruction that normally goes from I to E to be executed will go to I, E, and again to E if it is indirectly addressed.

Logic cycle (*L*). The L cycle is an execute cycle that does not require a reference to Mp. Many instructions use both E and L cycles when information is required from storage and the instruction cannot be completed during an E cycle. Other instructions require no reference to storage and, therefore, use only I and L cycles for their completion.

Buffer cycle (*B*). A buffer (B) cycle is a null Pc cycle; it is used when the data channels get information from or put information into core storage. This information can be either data or data-channel commands. All demands for B cycles come from the channels themselves. Because of the nature of Ms's and T's, the demand for a B cycle takes precedence over an instruction being performed by Pc. If Pc is in its logic cycle, then both an L and B cycle occur simultaneously.

Instruction interpretation. Instruction flow diagrams for the CLA, CAL, and CLS instructions are given in Fig. 4. These diagrams show the sequential process of instruction execution. Although the flow diagrams for these instructions are trivial, the general process is still apparent. The more complex instructions, for example, double-precision floating-point divide, are carried out in a similar fashion, but with many more operations. The registers, transfer paths, and interregister data operations are the register-transfer-level primitives from which the ISP is implemented. The data flow diagram (Fig. 3) explicitly defines the main registers and register operations within Pc.

Pc ISP

The Pc Instruction-set Processor is given in Appendix I of this chapter. The instructions are arranged in groups according to the location of operands. These groups are:

Operations on Mp

Mp ← u Mp	(*unary operation/u on Mp*)
Mp ← u Mps	(*unary operation on Mprocessor state/Mps*)
Mp ← Mp b Mps	(*binary operation/b*)

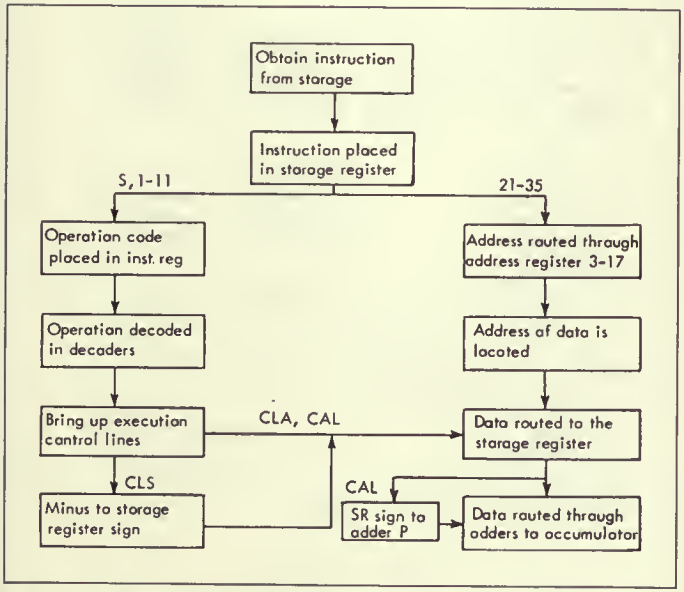


Fig. 4. IBM 7094 CLA and CLS instruction flowcharts. (*Courtesy of International Business Machines Corporation.*)

Operations on AC and MQ

Mps ← u Mps
Mps ← u Mp
Mps ← Mps b Mp

Operations on the index registers

Operations on the sense indicators

Instruction for program control

Memory mapping for multiprogramming and Mp(65536 w)

A special option provides multiprogramming by allowing a program to run in a protected area of Mp. Two registers are used: The base register establishes the lower bound of the program, and the length register establishes the upper bound. Pc checks that all program references are within the protected area.

Two Mp(32678 w)'s can be used on the computer. Mp is then considered as A core and B core for addresses 0:32767 and 32768:65535. A 1-bit register is used to select whether A or B core is to be used for data; and one I-bit register is used to select whether A or B core is to be used for the instruction. These modifications were used at M.I.T. in their Compatible Time Sharing System/CTSS [Corbato et al., 1962] which used a 7094 II.

Pio('7607 Data Channel)

The Pio('7607 Data Channel) executes programs which transfer data between Mp and Ms(magnetic tape) or T(card; reader, punch), (line; printer)). The paths and structure can be seen in Fig. 1.

Transferring blocks of data between Mp and an Ms or a T via the 7607 data channel takes places as follows:

1. Pc sets up the block transfer program in Mp for Pio.
2. Pc attaches a K for Ms(magnetic tape) or for T(card;reader) to Pio. (Faults in the connection may cause K to interrupt Pc.)
3. Pc starts the Pio by loading the Pio's instruction counter.
4. The data transmission takes place. On input, for example, T or Ms transmits a 6-bit character (or a 72-bit word) to K. The characters are buffered (collected) in K and sent on to Pio. Pio then requests a memory access from Mp via the S('7606 Multiplexor) and, finally, a data word is transmitted to Mp.
5. At the termination of a simple data block transfer, Pio fetches the next instruction from Mp. If the next instruction-task type is the same, Pio and K remain logically linked and continue to transmit data.

6 At the termination of the task, the completion signal from Pio causes Pc to interrupt and Pio may also halt.

Pio('IBM 7909 Data Channel)

Ms('1301 Disk Storage, '7340 Hypertape Drives) and the T('Tele-Processing equipment) communicate with Mp via the Pio('7909 Data Channel). Four 7909 Data Channels may be attached to a 7094 I or II system.

K('7631 File Control) is required for M(disks). Several K('7631) can be used with the 7094 system alone or shared with an IBM 1410 system or shared with another IBM 7000 series (not 7072 system).

When Ms('7340 Hypertape Drives) are attached to the 7094 system, K('7640 Hypertape Control) is used between the 7909 data channel and the drives. One K('7640) may be attached to a 7094 system; it has two paths, each of which can be used for data transmission.

The K('1416-6 Input-Output Synchronizer) is used with T('Teleprocessing Equipment)'s. The structure for these T's is rather elaborate, yet only six T's can be active at a time.

Transferring data from Mp to a T or an Ms via the 7909 takes place as follows:

1 Pc sets up the data-transfer management program in Mp for a Pio.

2 Pc starts Pio by setting Pio's command (instruction) location counter at the origin of the task program in Mp. (Faults in the connection may cause Pio interrupts to Pc.)

3 Pio issues an instruction to be executed by K. This establishes a state in K which selects and initializes the particular Ms or T and attaches the peripheral device K to Pio. (Faults in this selection may cause interruption of Pio.)

4 The data-transmission instruction is read and initializes Pio.

5 The data transmission takes place under control of Pio-K. The K of the selected device assembles characters. Input characters are transferred to Pio which assembles them into words and in turn transfers them to Mp.

6 At the termination of a data block transfer instruction, another instruction is fetched from Mp by Pio. This instruction may be to another K.

7 At the termination of the Pio program, Pio signals completion by interrupting Pc.

This discussion is based on information taken from the IBM 7094 Reference Manual. The body of the description is contained in ISP descriptions (Appendices 2, 3 and 4 of this chapter). The main registers of Pio are shown in Fig. 5. These registers are declared and their function is explained in the first section of the ISP description of Pio (Appendix 2). The remainder of the ISP description is concerned with defining the interpreter and the ISP instruction set.

There are about 50 bits in the K's (see Appendix 3). A knowledge of K's state and the K process is required for understanding the Pio. A description of the K and Pio data-transmission processes is given in Appendix 2.

The Pc instructions controlling Pio are presented in Appendix 4.

The level of detail in the appendices is slightly greater than that in normal ISP description. It is, however, not completely precise, as the behavior is extremely time- and Ms- or T-dependent. The sequence check conditions are incomplete; that is, the

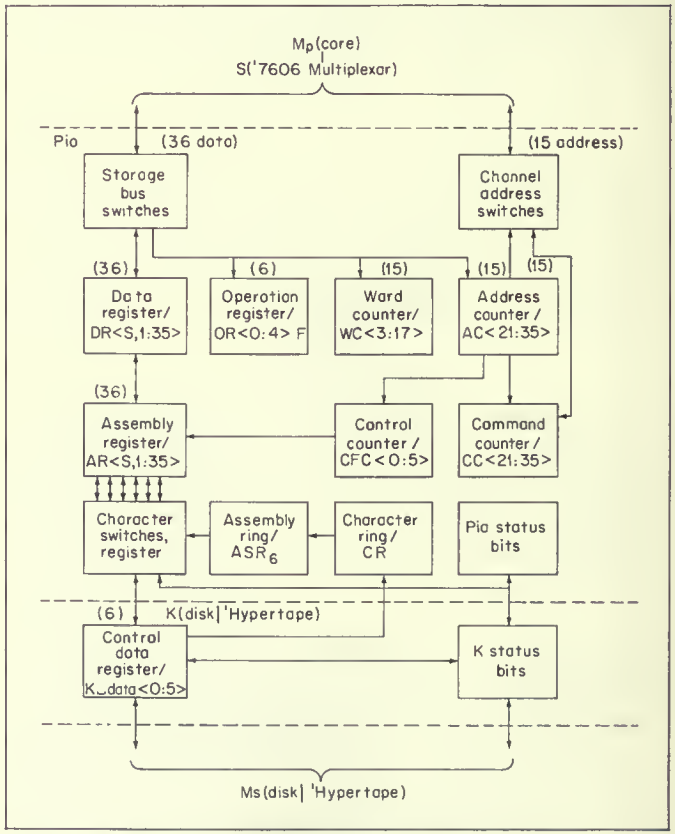


Fig. 5. IBM 7909 data-channel-registers diagram.

conditions for illegal instruction sequences are not given. Both ISP and text descriptions are given for parts which are particularly complex.

The ISP description should be observed in the following sequence: Pio State; K State (Appendix 3); Pio Instruction Format; Pio Interpreter; Pio Instruction—Control (or Initialization) instructions, Block Transfer (or Copy) instructions, Conventional Move and Transfer instructions, and Interrupt Control instructions; Instructions in Pc (Appendix 4); Interrupt Operation; and Processes defining data movements between K and Pio (Appendix 2). The Pio, K, and Ms or T processes are, in several ways, more complex than those of a Pc. First, Ms or T activity is not categorized as nicely as a Pc instruction set. The T or Ms events occur at times peculiar to the device—not a simple synchronous clock. Finally, the peripheral components have a large number of error states.

Conclusions

The series ending with the IBM 7094 II is a significant member of the computer population. It provides a good example of the evolution in computer systems that occurred from 1954 to 1965.

References

CorbF62; FrizC53; GreeJ57; GrumM58; RossH53; SaxoJ63; StevL52; A22–6703 IBM 7094 Principles of Operation

APPENDIX 1 IBM 7094 PC ISP

Appendix 1

IBM 7094 Pc ISP Description

Pc State

The description does not include the two protection and relocation schemes used for the 7040 and 7094. The Trap-Mode flip-flop is declared; its action is not described. Trap-Mode allows any change of the Instruction Counter to cause a trap. The Instruction Backup Register is not described, although it is used to save time in program execution. The description of the arithmetic functions is highly simplified.

```
AC<Q,P,S,1:35>                                  * Accumulator, 38 bits
   ACs<S,1:35>  := AC<S,1:35>                   * signed AC word
   AC1<P,1:35>  := AC<P,1:35>                   * logical AC word
   P := AC<P>                                   * carry for AC<1:35>; AC overflow is also set
   Q := AC<Q>                                   * carry for bits<P,1:35>
   S := AC<S>                                   * sign bit of AC
MQ<S,1:35>                                      * Multiplier-Quotient
ACMQ<S,Q,P,1:71>:= AC□MQ<1:35>                  * double word accumulator
SI<0:35>                                        Sense Indicators or program flags must be preserved if
                                                 double precision floating point instructions are given.
XR'[1:7]<3:17>                                  Index Registers in 7094
XR''[A,B,C]<3:17> := XR[1,2,4]<3:17>            * Index Registers for 704, 7090
Multiple␣Tag␣Mode                               program switch to force compatibility with 704, 7090; only
                                                 3 index registers XR[A,B,C] are in 704, 7090
IC<3:17>                                        * Instruction Location Counter
Run                                             * indicates whether machine is executing instructions
Divide␣Check                                    *
AC␣overflow                                     *
MQ␣overflow                                     *
Input␣Output␣check                              *
Trap␣request<A:H>                               Request to trap Pc from Pio #A...#H
Trap␣Mode                                       * Allows trapping or not of transfer instructions (not
                                                  described)
```

Pc Console State

```
Keys<0:35>                                      * console data
Sense␣Switches<0:5>
Sense␣Lights<0:3>
```

Mp State

```
M[0:32768-1]<S,1:35>                            Primary Memory of 2^15 w
```

Instruction Format

```
Instruction<S,1:35>                             corresponds to the physical Storage Register
   Y<21:35> := instruction<21:35>               generally the address part; used to calculate the effec-
                                                 tive address; corresponds to the physical Address Register
   T<18:20> := instruction<18:20>               the XR to use: 1,...7; 0 means no indexing; corresponds to
                                                 a physical register
   F<12:13> := instruction<12:13>               indirect address specification
      Indirect := (F<12:13> = 11)
   op<S,1:11>  := instruction<S,1:11>           op code; corresponds to a physical register
   hi␣op<0:2>  := instruction<S,1,2>            special op codes
```

* Denotes subset ISP, IBM 704, 7044 series

```
R<18;35>  := instruction<18;35>                    right half of instruction used to select SI bits
D<3:17>   := instruction<3:17>                     Decrement part of instruction, used to directly modify XR's
C'<12:17> := instruction<12:17>                    specifies variable length part of operation
C<10:17>  := instruction<10:17>                    convert instruction parameter
c<15:17>  := instruction<15:17>                    specifies character position in 7040, 7044 or extends op code

Effective Address Calculation Process
e<21:35> := (¬ indirect →e';                       effective address calculation
   indirect →instruction<18:35> ←M[e']<18:35>; next e')   1 level indirect addressing
e'<21:35>:= ((T = 0) →Y;                           indexed effective
             (T ≠ 0) →Y+XR[T] )
e''<23:35>:= e'<23:35>
sc<28:35>:= e'<28:35>                              a truncation of e, used for specifying number of shifts:
                                                    corresponds to a physical register
XR[T]<3:17> := (                                   index registers are or'd together in multiple tag mode
   ¬Multiple_Tag_Mode →XR'[T];
    Multiple_Tag_Mode →(
     (T<13>→XR''[A]) ∨ (T<19> →XR''[B]) ∨ (T<20> →XR''[C])))
```

The description for Multiple Tag Mode is incomplete for the case of writing in several index registers at one time. The only way this could be accomplished in the description would be to define each load index register instruction as microprogrammed.

```
Data Formats
sl<S,1:35>                                         logical data; unsigned integer/boolean vector
sx<S,1:35>                                         single precision fixed point (integer) data
   sx sign := sx<S>
   sx magnitude<1:35> := sx<1:35>
sf<S,1:35>                                         single precision floating point value of: sf_sign□sf_mantissa
   sf sign := sf<S>                                 x2^sf_exponent
   sf exponent<1:8> := 200(8) - sf<1:8>
   sf mantissa<0:26>:= sf<9:35>
df[0:1]<S,1:35>                                    double precision floating point value of: df_sign□df_mantissa
   df sign := df[0]<S>
   df exponent<1:8> := 200(8) - df[0]<1:8>          x2^df_exponent
   df mantissa<0:53>:= df[0:1]<9:35>

Instruction Interpretation Process
Run →(instruction ←M[IC]; IC ←IC+1; next           fetch
      instruction_execution)                       execute

Instruction Set and Instruction Execution Process
Instruction_execution := (
Operations on M: M[e] ← f; or M[e] ← f(M[e]);
   STZ (:= op = 600) →M[e] ←0;                     * store zero
   MSP (:= (op = -1623) ∧ (c = 7)) →M[e]<S> ←0);   make sign positive; 704 series only
   MSM (:= (op = -1623) ∧ (c = 6)) →M[e]<S> ←1);   make sign minus; 704 series only
Block transfer of data, M ← M (704 series only)
   TMT (:= op = -1704) →(M[AC<21:35>:·(AC<21:35> + e'<28:35>)] ←
                         M[AC<3:17>:(AC<3:17> + e'<28:35>)]);
```

```
Single word data transmission to M, M[e] ← Register
   STQ (:= op = -600) → (M[e] ← MQ);                          * store MQ
   SLQ (:= op = -620) → (M[e]<S,1:17> ← MQ<S,1:17>);           * store left half MQ
   STO (:= op = 601) → (M[e] ← ACs);                          * store
   SLW (:= op = 602) →(M[e] ← AC1);                           * store logical word
   STP (:= op = 630) →(M[e]<S,1,2> ← AC<P,1,2>);              * store prefix
   STD (:= op = 622) →(M[e]<3:17> ← AC<3:17>);                * store decrement
   STT (:= op = 625) → (M[e]<18:20> ← AC<18:20>);             * store tag
   STA (:= op = 621) → (M[e]<21:35> ←AC<21:35>);              * store address
   STL (:= op = -625) → (M[e]<21:35> ← IC);                   store instruction location counter
   STR (:= hi_op = -1) → (M[0]<21:35> ← IC; IC ←2);           store instruction location counter and trap
   STI (:= op = 604) → (M[e] ←SI);                            store indicators
Double length data transmission to M from A
   DST (:= op = -603) → (M[e]□M[e+1] ←ACs□MQ);                double store
Binary operation with AC: M[e] ← AC b M[e];
   ORS (:= op = -602) → (M[e] ←AC1 ∨ M[e]);                   * or to storage
   ANS (:= op = 320) → (M[e] ←AC1 ∧ M[e]);                    * and to storage
6 bit character to M from AC, (7040 only);
   SAC (:= op = -1623) → (M[e]<c x 6 :(c x 6+5)> ← AC<30:35>);
Operations to the AC,MQ, or AC□MQ with AC,MQ,ACMQ, Keys and M operands:
   CLM (:= (op = 760) ∧ (e' = 0)) → (AC<Q,P,1:35> ←0);        clear magnitude
   SSP (:= (op = 760) ∧ (e' = 3)) → (AC<S> ← 0);              * set sign plus
   SSM (:= (op = -760) ∧ (e' = 3)) → (AC<S> ← 1);             * set sign minus
   CLA (:= op = 500) → (AC ←0; next ACs ←AC+M[e]);            clear and add
   CAL  (:= op = -500) → (AC ←0; next AC1 ←AC1+M[e]);         clear and add logical
   CLS (:= op = 502) → (AC ←0; next AC ←AC-M[e]);             clear and subtract
   LDQ (:= op = 560) → (MQ ←M[e]);                            load MQ
   ENK (:= (op = 760) ∧ (e' = 4)) → (MQ ←Keys);               enter Keys
   PIA (:= op = -46) → (AC ←SI);                              place indicators in AC
   DLD (:= op = 443) → (ACs□MQ ← M[e]□M[e+1]);                double load
Operations with AC,AC ← f(AC)
   CHS (:= (op = 760) ∧ (e' = 2)) → (AC<S> ←¬ AC<S>);         change sign
   COM (:= (op = 760) ∧ (e' = 6)) → (AC<Q,P,1:35> ←¬ AC<Q,P,1:35>);  *complement magnitude
   RND (:= op = 760) ∧ (e' = 10)) →MQ<1> →AC ←AC + 1;         * round
   FRN (:= op = 760) ∧ (e' = 11)) → (AC ← round(ACMQ){sf});   * floating round
   ALS (:= op = 767) → (AC<Q,P,1:35> ←AC<Q,P,1:35> x 2^SC);   * AC left shift
   ARS (:= op = 771) → (AC<Q,P,1:35> ←AC<Q,P,1:35> / 2^SC);   * AC right shift
   LLS (:= op = 763) → (ACMQ' ←ACMQ' x 2^SC);                 * long left shift
   LRS (:= op = 765) → (ACMQ' ←ACMQ'/ 2^SC);                  * long right shift
      ACMQ'<0:71> := AC<Q,P,1:35> □ MQ<1:35>
   LGL (:= op = -763) → (ACMQ'' ←ACMQ'' x 2^SC {logical});    * logical left shift
   LGR (:= op = -765) → (ACMQ'' ←ACMQ''/ 2^SC {logical});     * logical right shift
      ACMQ''<0:72> := AC<Q,P,1:35> □ MQ<S,1:35>
   RQL (:= op = -773) → (MQ ←MQ x 2^SC {rotate});             * rotate MQ left
Exchange of Data between registers, AC, and MQ
   XCA (:= op = 131) → (AC ←MQ; MQ ←AC);                      exchange AC and MQ
```

```
    XCL (:= op = -130) → (MQ ←AC1; AC1 ←MQ; AC<S,Q> ←0);          exchange logical AC and MQ
  6 bit character to AC from M (704 only)
    PCS (:= op = -1505) → (AC<3D:35> ←M[e ]<(c x 6): (c x 6 + 5)>);place character from storage
  Binary operations with M,AC← AC b M;
    ADD (:= op = 400) → (AC ←AC + M[e]);                          * add
    ADM (:= op = 401) → (AC ←AC + abs(M[e]));                     * add magnitude
    SUB (:= op = 4D2) → (AC ←AC - M[e] );                         * subtract
    SBM (:= op = -400) → (AC ←AC - abs(M[e]));                    * subtract magnitude
    MPY (:= op = 200) → (ACMQ ←MQ x M[e]; AC<Q,P> ←0);            * multiply
    MPR (:= op = -200) → (ACMQ ←MQ x M[e]; next                   * multiply and round
                          MQ<1> →AC ←AC + 1; AC<Q,P> ←0);
    DVH (:= op = 220) → (AC,MQ ←ACMQ / M[e]; next
                         Divide_check →Run ←0);                   * divide or halt
    DVP (:= op = 221) → (AC,MQ ←ACMQ / M[e]);                     * divide or proceed; Divide_check may be set
    ACL (:= op = 361) → (AC1 ←AC1 + M[e]);                        * add and carry logical word
The following are variable length x and / operations.  C' specifies the length of divisor or multiplier.
    VLM (:= op = 2D4) → (ACMQ ←MQ x M[e] {v1});                   variable length multiply
    VDP (:= op = 225) → (AC,MQ ←ACMQ / M[e] {v1});                variable length divide or proceed
    VDH (:= op = 224) → (AC,MQ ←ACMQ / M[e] {v1}; next            variable length divide or halt
                         Divide_check →Run ←0);
  Single precision floating point
    FAD (:= op = 30D) → (AC,MQ ←AC + M[e] {sf});                  * add
    FAM (:= op = 304) → (AC,MQ ←AC + abs(M[e]) {sf});             * add magnitude
    FSB (:= op = 3D2) → (AC,MQ ←AC - M[e] {sf});                  * subtract
    FSM (:= op = 3D6) → (AC,MQ ←AC - abs(M[e]) {sf});             * subtract magnitude
    FMP (:= op = 260) → (AC,MQ ←MQ X M[e] {sf});                  * multiply
    FDH (:= op = 24D) → (AC,MQ ←AC / M[e] {sf}: next              * divide or halt
                         Divide_check →Run ←D);
    FDP (:= op = 241) → (AC,MQ ←AC / M[e] {sf});                  * divids or proceed
  Unnormalized single precision floating point
    UFA (:= op = -30D) → (AC,MQ ←AC + M[e] {suf});                * add
    UAM (:= op = -3D4) → (AC,MQ ←AC + abs(M[e]) {suf});           * add magnitude
    UFS (:= op = -3D2) → (AC,MQ ←AC - M[e] {suf});                * subtract
    USM (:= op = -3D6) → (AC,MQ ←AC - abs(M[e]) {suf});           * subtract magnitude
    UFM (:= op = -260) → (AC,MQ ←MQ x M[e] {suf});                * multiply
  Double precision floating point
   In DF operations, the SI are used as temporary registers and will be changed.
    DFAD (:= op = 301) → (                                        *add
       ACMQ ←ACMQ + M[e]□M[e+1] {df}; SI ←?);
    DFAM (:= op = 3D5) → (                                        * add magnitude
       ACMQ ← ACMQ + abs(M[e]□M[e+1]) {df}; SI ← ?);
    DFSB (:= op = 303) → (                                        * subtract
       ACMQ ←ACMQ - M[e]□M[e+1] {df}; SI ←?);
    DFSM (:= op = 307) → (                                        * subtract magnitude
       ACMQ ←ACMQ - abs(M[e]□M[e+1]) {df}: SI ←?);
```

```
    DFMP (:= op = 261) → (                                  * multiply
       ACMQ ← ACMQ x M[e]□M[e+1] {df}; SI ← ?);
    DFDM (:= op = -240) → (                                 * divide or halt
       ACMQ ← ACMQ / M[e]□M[e+1] {df}; SI ← ?; next
       Divide_check → Run ← 0);
    DFDP (:= op = -241) → (                                 * divide or proceed; Divide check may be set
       ACMQ ← ACMQ / M[e]□M[e+1] {df}; SI ← ?);
Unnormalized double precision floating point
    DUFA (:= op = -301) → (                                 * add
       ACMQ ← ACMQ + M[e]□M[e+1] {duf}; SI ← ?);
    DUAM (:= op = -305) → (                                 * add magnitude
       ACMQ ← ACMQ + abs(M[e]□M[e+1]){undf}; SI ← ?);
    DUFS (:= op = -303) → (                                 * subtract
       ACMQ ← ACMQ - M[e]□M[e+1] {duf}; SI ← ?);
    DUSM (:= op = -307) → (                                 * subtract magnitude
       ACMQ ← ACMQ - abs(M[e]□M[e+1]){duf}; SI ← ?);
    DUFM (:= op = -261) → (                                 * multiply
       ACMQ ← ACMQ x M[e]□M[e+1] {duf}; SI ← ?);
Logical
    ORA (:= op = -501) → (AC1 ← AC1 ∨ M[e]);                * or to accumulator
    ANA (:= op = -320) → (AC1 ← AC1 ∧ M[e]);                * and to accumulator
    ERA (:= op = 322) → (AC1 ← AC1 ⊕ M[e]);                 * exclusive or to accumulator
```

The convert instructions are not described in detail. These instructions take a table in memory, addressed by the 6, 6 bit characters in AC or MQ and form a sum of products in the AC or MQ for each character component of the word.

```
    CVR (:= op = 114) → (AC,MQ ← f(AC,C,XR[1],M[Y:Y+63]));      convert by replacement from the AC
    CRQ (:= op = -154) → (AC,MQ ← f(MQ,C,XR[1],M[Y:Y+63]));     convert by replacement from the MQ
    CAQ (:= op = -114) → (AC,MQ ← f(AC,MQ,C,XR[1],M[Y:Y+63]));  convert by addition from the MQ
Transmission between M,XR[T], and AC
 If tag,T,=0, then a no operation occure
    PDX (:= op = -734) → (XR[T] ← AC<3:17>);                * place decrement in index
    PAX (:= op = 734) → (XR[T] ← AC<21:35>);                * place address in index
    PDC (:= op = -737) → (XR[T] ← 2^15 - AC<3:17>);         * place complement of decrement in index
    PAC (:= op = 737) → (XR[T] ← 2^15 - AC<21:35>);         * place complement of address in index
    LXD (:= op = -534) → (XR[T] ← M[Y]<3:17>);              * load index from decrement
    LXA (:= op = 534) → (XR[T] ← M[Y]<21:35>);              * load index from address
    LDC (:= op = -535) → (XR[T] ← 2^15 - M[Y]<3:17>);       * load complement of decrement in index
    LAC (:= op = 535) → (XR[T] ← 2^15 - M[Y]<21:35>);       * load complement of address in index
    AXT (:= op = 774) → (XR[T] ← Y);                        * address to index true
    AXC (:= op = -774) → (XR[T] ← 2^15 - Y);                * address to index complement
    PXD (:= op = -754) → (AC ← 0; next AC<3:17> ← XR[T]);   * place index in decrement
    PXA (:= op = 754) → (AC ← 0; next AC<21:35> ← XR[T]);   * place index in address
    PCD (:= op = -756) → (AC ← 0; next AC<3:17> ← 2^15 - XR[T]);  * place complement of index in decrement
    PCA (:= op = 756) → (AC ← 0; next AC<21:35> ← 2^15 - XR[T]);  * place complement of index in address
    SXD (:= op = -634) → (M[Y]<3:17> ← XR[T]);              * store index in decrement
    SXA (:= op = 634) → (M[Y]<21:35> ← XR[T]);              * store index in address
```

```
   SCO (:= op = -636) → (M[Y]<3:17> ← 2^15 - XR[T]);          * store complement of index in decrement
   SCA (:= op = 636) → (M[Y]<21:35> ← 2^15 - XR[T]);          * store complement of index in address
Transmission to Sense Indicators
   PAI (:= op = 44) → (SI ← AC1);                             place accumulator in indicators
   LDI (:= op = 441) → (SI ← M[e]);                           load indicators
   OAI (:= op = 43) → (SI ← SI ∨ AC1);                        or accumulator to indicators
   RIA (:= op = -42) → (SI ← SI ∧ ¬ AC1);                     reset indicators from accumulator
   IIA (:= op = 41) → (SI ← SI ⊕ AC1);                        invert indicators from accumulator
   OSI (:= op = 442) → (SI ← SI ∨ M[e]);                      or storage to indicators
   RIS (:= op = 445) → (SI ← SI ∧ ¬ M[e]);                    reset indicators from storage
   IIS (:= op = 440) → (SI ← SI ⊕ M[e]);                      invert indicators from storage
   SIL (:= op = -55) → (SI<0:17> ← SI<0:17> ∨ R);             set indicators of left half
   RIL (:= op = -57) → (SI<0:17> ← SI<0:17> ∧ ¬ R);           reset indicators of left half
   IIL (:= op = -51) → (SI<0:17> ← SI<0:17> ⊕ R);             invert indicators of left half
   SIR (:= op = 55) → (SI<18:35> ← SI<18:35> ∨ R);            set indicators of right half
   RIR (:= op = 57) → (SI<18:35> ← SI<18:35> ∧ ¬ R);          reset indicators of right half
   IIR (:= op = 51) → (SI<18:35> ← SI<18:35> ⊕ R);            invert indicators of right half
Program flow control instructions
   NOP (:= op = 761) → ;                                      no operation
   HPR (:= op = 420) → (Run ← 0);                             * halt and proceed
   HTR (:= op = 0) → (Run ← 0; IC ← e);                       * halt and transfer
   TRA (:= op = 20) → (IC ← e);                               * transfer
   XEC (:= op = 522) → (instruction ← M[e]; next              execute
      Instruction_execution);
Conditional transfers
   TZE (:= op = 100) → ((AC<Q,P,1:35> = 0) → IC ← e);         * transfer on zero
   TNZ (:= op = -100) → (¬ (AC<Q,P,1:35> = 0) → IC ← e);      * transfer on no zero
   TPL (:= op = 120) → (¬ AC<S> → IC ← e);                    * transfer on plus
   TMI (:= op = -120) → (AC<S> → IC ← e);                     * transfer on minus
   TOV (:= op = 140) → (AC_overflow → IC ← e;                 * transfer on overflow
                        AC_overflow ← 0);
   TNO (:= op = -140) → (¬ AC_overflow → IC ← e;              * transfer on no overflow
                         AC_overflow ← 0);
   TQP (:= op = 162) → (¬ MQ<S> → IC ← e);                    * transfer on MQ plus
   TQO (:= op = 161) → (MQ_overflow → IC ← e;                 * transfer on MQ overflow
                        MQ_overflow ← 0);
   TLQ (:= op = 40) → ((AC > MQ) → IC ← e);                   * transfer on low MQ
   TIO (:= op = 42) → ((AC1 = (AC1 ∧ SI)) → IC ← e);          * transfer when indicators on
   TIF (:= op = 46) → ((0 = (AC1 ∧ SI)) → IC ← e);            * transfer when indicators off
Index manipulation and control and subroutine calling
   TSX (:= op = 74) → (XR[T] ← 2^15 - IC; IC ← Y);            * transfer and set index
   TSL (:= op = -1627) → (M[e]<21:35> ← IC; IC ← e + 1);      * 704
Loop control
   TXI (:= hi_op = 1) → (XR[T] ← XR[T] + 0; IC ← Y);          * transfer with index incremented
   TXH (:= hi_op = 3) → ((0 < XR[T]) → IC ← Y);               * transfer on index high
```

```
   TXL (:= hi_op = -3) → ((D ≥ XR[T]) → IC ← Y);                  * transfer on index low or equal
   TIX (:= hi_op = 2) → ((XR[T] > D) → (XR[T] ← XR[T] - D;        * transfer on index
                         IC ← Y));
   TNX (:= hi_op = -2) → ((XR[T] > D) → XR[T] ← XR[T] - D;        * transfer on no index
                         (XR[T] ≤ D) → IC ← Y);
Skip tests
   MIT (:= (op = -1341) ∧ (c = 7)) → (M[e]<S> → IC ← IC + 1);     storage minus test; 704 series only
   PLT (:= (op = -1341) ∧ (c = 6)) → (¬ M[e]<S> → IC ← IC + 1);   storage plus test; 704 series only
   CCS (:= ((op = -1341) ∧ (c < 6)) → (                           compare
      (AC<30:35> = M[e]<(c x 6):(c x 6 + 5)>) → IC ← IC + 1;      character with storage; 704 series only
      (AC<30:35> < M[e]<(c x 6):(c x 6 + 5)>) → IC ← IC + 2));
   PBT (:= (op = -760) ∧ (e'' = 1)) → (AC<P> → IC ← IC + 1);      * P bit test
   DCT (:= (op = +760) ∧ (e'' = 12)) → (Divide_check → IC ← IC + 1)* Divide_check test
   LBT (:= (op = +760) ∧ (e'' = 1)) → (AC<35> → IC ← IC + 1);     * low bit test
   ZET (:= op = +520) → ((M[e] = 0) → IC ← IC + 1);               * storage zero test
   NZT (:= op = -520) → ((M[e] ≠ 0) → IC ← IC + 1);               * storage own zero test
   CAS (:= op = +340) → (                                         * compare AC with storage
      (ACs = M[e]) → IC ← IC + 1;
      (ACs < M[e]) → IC ← IC + 2);
   LAS (:= op = -340) → (                                         * logical compare AC with storage
      (AC<Q,P,1:35> = M[e]<S,1:35>) → (IC ← IC + 1):
      (AC<Q,P,1:35> < M[e]<S,1:35>) → (IC ← IC + 2));
   SWT (:= (op = 760) ∧ (e'<9:14> = 16)) → (                      Sense_Switches test
      Sense_Switches<e'<15:17>> → IC ← IC + 1);
   SLF (:= (op = 760) ∧ (e' = 140)) → (Sense_Lights<0:3> ← 0);    Sense_lights off
   SLN (:= (op = 760) ∧ (e'<9:14> = 14) ∧ (e'<15:17> ≠ 0)) → (    Sense_lights on
      Sense_Lights<e'<15:17>> ← 1);
   SLT (:= (op = -760) ∧ (e'<9:14> = 14)) → (                     Sense_lights test
      Sense_Lights<e'<15:17>> → (IC ← IC + 1; Sense_Lights<e'<15:17>> ← 0));
   ETM (:= (op = 760) ∧ (e' = 7)) → (Trap_Mode ← 1);              enter Trap_Mode
   LTM (:= (op = -760) ∧ (e' = 7)) → (Trap_Mode ← 0);             leave Trap_Mode
   EMTM (:= (op = -760) ∧ (e' = 16)) → (Multiple_Tag_Mode ← 1);   enter Multiple_Tag_Mode
   LMTM (:= (op = 760) ∧ (e' = 16)) → (Multiple_Tag_Mode ← 0);    leave Multiple_Tag_Mode
                  )                 end Instruction_execution
```

APPENDIX 2 IBM 7909 DATA CHANNEL ISP DESCRIPTION (A PIO)

Appendix 2

IBM 7909 Data Channel ISP Description (a Pio)

Although the following description is of a Pio, signals generated in Pc, M, and K are necessary. Appendices 1, 3, and 4 are also necessary for a complete description. The Ms attached to K controls the precise time information flows.

Pio State

CC<21:35>	*Command Counter; 15 bit command (or instruction) counter containing the location of the next command*
AC<21:35>	*Address Counter; during vector data transfers AC contains the address of the next data word to transfer. During a transfer command AC is set to the address of the next command*
AR<S,1:35>	*Assemblu Register; a buffer for data flow between the data register and the device control registers*
ARc[0:5]<0:5> := AR<S,1:35>	*character array defined by AR; a character is normally selected ARc[ASR]*
CTC<0:5>	*Control Counter; a 6 bit register which can be loaded and stored by the ISP*
WC<3:17>	*Word Counter; a counter controlling the number of words left to transfer during a command*

Data transmission modes in Pio for the K-Pio dialogue;
These control the flow direction and data types between K and Pio. Although not described as such, each indicator is mutually exclusive of the others.

SNI	*Sense Indicator; K is transmitting sense data to Pio.*
WRI	*Write Indicator: K is receiving data from Pio.*
RDI	*Read Indicator; K is transmitting data to Pio.*
Wait	*bit denotes a halted condition in Pio; instructions are not executed*
IL := 42_8	*Interrupt Location for Pio #A to interrupt itself. Each of the 8 Pio's have special locations. two locations, IL, IL+1, are reserved*
Interrupt_Request := ((CKC<1:6> ∧ CKCI<30:35>) ≠ 0)	*signifies a request to interrupt Pio from K or within Pio*
Pc_Trap_Request	*signifies a request to trap Pc from Pio*
Interrupt Mode	*bit to denote that an interrupt program is running in Pio*
CKC<1:6>	*Check Conditions in K that cause an interrupt of the Pio*
CKC<1>/Input_Output_Check/I_O_Check	
CKC<2>/Sequence_Check	
CKC<3>/K_Unusual_End	
CKC<4:5>/Attention Conditions<1:2>	
CKC<6>/K_Check	
CKCI<30:35>	*a mask to inhibit Pio interrupts from CKC*

The CKC indicators are described as follows;

Input_Output_Check

This condition occurs when the channel fails to obtain a storage reference cycle in time to satisfy demands of the attached IO device. The condition is also monitored in the Pc. I_O_Check is turned off when an LIP or LIPT command is executed or when the Pc executes an RSC or RIC instruction.

When an I_O_Check occurs, the adapter is disconnected and an interrupt occurs when the K_End signal is received from the adapter (K). The command counter contains the location plus one of the present command. The address counter contains the location plus one or two of the last word transmitted if the operation was a write or control, or the location plus one of the last word transmitted if the operation was a read or sense.

If an I_O_Check occurs while the channel is in interrupt mode, the I_O_Check is not recognized and is not saved.

Sequence_Check

A Sequence_Check indicates an invalid sequence of channel commands. If a Sequence_Check occurs during data transmission, the adapter is logically disconnected and the interrupt occurs when the K_End signal is received.

The following instructions cause a Sequence_Check and a channel interrupt. (The checks are not described in the ISP description.)

1. *If a CTLW, CTLR, or SNS is followed by CTL, CTLW, WTR, TWT, or SNS.*
2. *If an SNS or CPYP is followed by any command other than a CPYP, CPYD, TCH, or TDC.*
3. *If a TCH or TDC following an SNS or CPYP transfers control to any command other than a CPYP, CPYD, TCH, or TDC.*
4. *If a CPYP or CPYD has not been properly preceded by a CTLW, CTLR, or SNS.*

K_Unusual_End

This signal indicates an error condition recognized by K. It causes an immediate interrupt to Pio. The signal may be determined by sensing the K error indication.

Attention Conditions

This is a signal indicating a change in status of the attached input output device. For example, during disk operations, an attention signal is generated when an access mechanism has completed a seek operation. The particular access mechanism that generated this indication may be determined from sense data.

K_Check

Adapter check (K_Check) indicates an error and is recognized by the 7909, but does not necessarily indicate a K malfunction. The conditions which cause an adapter check are:

1. *Circuit failure occurs in the ASR or CR.*
2. *The character rate of the attached IO device exceeds the capability of the channel.*
3. *The adapter (K) is not operational. This indication occurs if power is off on the adapter and an attempt is made to read, write, control or sense.*

Hardware Switches

These gates route information among the registers on a selected basis. They are not under control of the program and are not registers.

Storage Bus Switches<S,1:35>	*These 36 switches (and/or gates) provide the data path to and from the 7606 Multiplexor for data or command entry into the Pio.*
Channel Address Switches<21:35>	*These 15 switches provide the Mp with address information. Address information is selected from the Address Counter or the Command Counter.*
Character Switches<0:5>	*These 6 bit switches enable the character to be read from or written into the Assembly Register.*

Pio State (not in ISP)

Hardware registers not in ISP but used in the description and the Pio.

OR<0:4>	*Operation Register. The register containing the operation part of the instruction. OR is made up from i<S,1:3,19>.*
DR<S,1:35>	*Data Register. A buffer for data flow between M and the AR.*
CR	*Character Ring. A register to control the timing or transmission into AR.*
ASR_6	*Assembly Ring. The counter to control the gates to/from AR from/boK. Data are sent to or received from the control, Y, one 6-bit character at a time via the Character Switches under control of ASR.*

Instruction Format

i<S,1:35>	*instruction: normally IBM calls these commands because a Pio executes them*
f := i<18>	*indirect*
op<0:4> := i<S,1:3,19>	*operation code*
y<0:14> := i<21:35>	*address*
c<0:14> := i<3:17>	*count part*
c'<0:2> := i<3:5>	
m<0:5> := i<12:17>	*mask*

```
   e<21;35>  := (¬ f →y; f →M[y]<21;35>);                    1 level of indirect addressing

Mp State
   M[0:32768-1]<S,1:35>                                      Computer's primary memory

Instruction Interpretation Process
   ¬ Interrupt_request ∧ ¬ Wait →(Instruction ←M[CC];        fetch, no interrupt
        CC ←CC+1; next
        Instruction_execution);                              execute, no interrupt
     Interrupt_request ∧ ¬ Interrupt_mode →(                 interrupt process
        (M[IL]<21:35> ← CC;M[IL]<3:17> ←CC;
           Interrupt_mode ←1; next CC ←IL+1);
```

Pio Interrupts and Pc Traps

The Pio is capable of having its stored program interrupted independently of other P's. This operation is separate and distinct from a data channel trap in which Pio interrupts the Pc. On recognition of an interrupt condition the Pio stores the contents of the command and address counters in a fixed memory location, IL, and then executes the command located in the next location.

If the 7909 channel is to be diverted from normal command execution sequence, the command in the fixed location must be one that will change the contents of the command counter (TCH, LIPT, or successful TDC or TCM). If this command is other than a successful transfer, the channel executes it and resumes operation at the location immediately following the location where the interrupt occurred. If the command at the fixed location is a WTR or TWT, the channel suspends operation as described in the channel command section, but the command counter contains the location plus one of the command responsible for the interrupt.

Interrupt conditions are stored in a six-position register in the data channel and may be examined with the TCM command. Any combination of interrupt conditions causes an interrupt; however, once interrupted the channel is placed in interrupt mode and further attempts to set the interrupt condition or to interrupt are inhibited. The channel remains in interrupt mode until an LIP or LIPT command is executed by the channel or an RIC instruction is executed by the CPU. If a channel is in interrupt mode and an RSC instruction is executed by the CPU before the channel executes a LIP or LIPT command, the interrupt condition register is reset but the channel remains in interrupt mode. An LIP or LIPT command or a RIC instruction is the only program means available to cause the channel to exit from interrupt mode and become receptive to further interrupt conditions.

Interrupts are also inhibited if channel trap is in process on that channel. This inhibiting persists until either an RSC or STC instruction (depending on whether the channel was enabled) is executed by the Pc.

This command, when decoded by a channel not prepared to read or write, causes a sequence check and, thus, a channel interrupt. If the channel is prepared to read or write, this command causes c words to be transmitted between the channel and Mp, starting with M[e]. Data transmission continues until c is reduced to zero or a K_End signal is received by the channel. In either case, the channel read or write indicator is reset. If, while a CPYD is being executed a K_End signal is received before the count is reduced to zero, the channel read or write indicator is reset, and the channel obtains a new command from the next sequential location.

If the next command is other than a copy, the channel executes that command. If the next command is a copy, the channel interrupts on a program sequence check. The last word transmitted to storage under CPYD control remains in the assembly register if a K_End signal is received before the word count reaches zero.

If the count for the CPYD goes to zero before the K_End signal is received, the channel initiates a disconnect but does not get the next sequential command until a K_End or K_Unusual_End signal is obtained. In general, when opsrating under CPYD control, the channel does not obtain the next sequential command until either a K_End or a K_Unusual_End signal causes an interrupt.

Instruction Set and Instruction Execution Process

The following control commands transmit instructions (orders) or operation information to K. Information is sent to K from M[e] starting with the high order 6 bit character and continues until a K_End is received by Pio from K. If more than one control word is required, the next words come from M[e+1,e+2,...].

For CTL, CTLR, and CTLW instructions, the control words are first transmitted. Next the Read or Write indicator is set in Pio.

```
   Instruction execution := (
   CTL  (:= op = 01000) →(AC ←e;                             control
                          Move_word_from_M; ASR ←0; next
                          Move_control_char_to_K);
   CTLR (:= op = 01001) →(AC ←e;                             control and read
                          Move_word_from_M; ASR ←0· next
                          Move_control_char_to_K; RDI ←1);
   CTLW (:= op = 01010) →(AC ←e; next                        control and write
                          Move_word_from_M; ASR ←0· next
                          Move_control_char_to_K; WRI ←1);
```

```
CPYD (:= op = 101$01) → (AC ←e;                          copy and disconnect
                         Copy_data_block; next
                         RDI□SNI□WRI ←0; K_end_wait);
CPYP (:= op = 100$0)   (AC ←e;                           copy and proceed
                         Copy_data_block);
SNS  (:= op = 01011) → (SNI ← 1);                        sense
Execution of this command must be followed by a copy command. The data in K's sense indicators are sent via the K_Data
register through AR and DR to M.
SMS  (:= (op = 11100) ∧ (c'=0)) →CKCI ←e<29:35>:         set mode and select
LCC  (:= op = 11011) → (AC ←e; next                      load control counter
                        CTC ←AC<30:35>):
TDC  (:= op = 11010) → (AC ←e; next                      transfer and decrement counter
   (CTC = 0) → :
   (CTC ≠ 0) → (CTC ←CTC-1; CC ←AC));
ICC  (:= op = 111$1) → (                                 insert control counter
   (0 < c' < 7) →ARc[c']← CTC;
   (c' = 0) →ARc[5] ←CKCI;
   (c' = 7) → :);
TCM  (:= op = 101$1) → (                                 transfer on conditions met
   ((c' = 0) ∧ ¬ i<11> ∧ (m = CKC)) → (CC ←e);
   ((c' = 0) ∧ i<11> ∧ ((m ∧ CKC) = m)) → (CC ←e);
   ((0 < c' < 7) ∧ ¬ i<11> ∧ (m = ARc[c])) → (CC ←e);
   ((0 < c' < 7) ∧ i<11> ∧ ((m ∧ ARc[C])  = m)) → (CC ←e);
   ((c' = 7) ∧ (m = 0)) → (CC ←e));
TCH  (:= op = 001$0) → (CC ←e);                          transfer in channel
LAR  (:= op = 01100) →(AC ←e; next AR ←M[AC]);           load assembly register
SAR  (:= op = 01101) →(AC ←e; next M[AC] ←AR);           store assembly register
XMT  (:= op = 00011) →(AC ←e; WC ←c; next                an instruction to move c words in M[CC:(CC + c)] to M[e:(e + c)]
                        M_block_move)
XMT is actually a vector move within Mp.
XMT  (:= op = 000$1) →((c ≠ 0) →(                        vector move
                          M[e:(e + c - 1)] ←M[CC:(CC + c - 1)];
                          WC ←0; AC ←AC + c;             fix end conditions
                          CC ←CC + c)):
WTR  (:= op = 000$0) →(AC ←e; Wait ←1);                  wait and transfer
TWT  (:= op = 01110) →(AC ←e; Wait ←1;                   trap and wait
                        Pc_Trap_Request ←1):
LIP  (:= op = 11001) → (                                 leave interrupt program
    CC ←M[IL]<21:35>;
   CKC ←0; Interrupt_Mode ←0);
LIPT (:= 001$1) → (                                      leave interrupt program and transfer
   (CC ←e; CKC ←0;
    Interrupt_Mode ←0)
          )                                              end Instruction_execution
```

```
K, Pio, and M Data Movement Processes
The following processes define the movement of characters and words among the registers and Memory.  The principle activity is
copy_data_block.  On writing, a word is taken from M and placed in Pio, then transferred character by character to K.  On
reading, a character is taken from K and assembled in Pio, then transferred as a word to M.  The following processes move
either characters or words in a direction relative to Pio.

  Move_char_to_K                                          writing into K
  Move_control_char_to_K                                  setting up instruction in K
  Move_char_from_K                                        reading from K
  Move_word_to_M                                          writing into M
  Move_word_from_M                                        reading from M
  M_block_move                                            read M, write M on a word by word basis
  K_end_wait                                              process to wait for K end signals
  Copy_data_block := (
     RDI → (Move_char_from_K; ASR ← 0);
     SNI → (Move_char_from_K; ASR ← 0);
     WRI → (Move_word_from_M; ASR ← 0; WC ← WC - 1; next
        Move_char_to_K))
  Move_char_to_K := (                                     K ← Pio ← M data movement
      K_End ∨ (WC = 0) → ;                                stop at end
      ¬ K_End  ∧ (WC ≠ 0) ∧ K_Data_Rq → (                 transmit a char
        (ASR = 0) → Move_word_from_M; WC ← WC-1; next
        K_Data ← ARc[ASR]; ASR ← ASR + 1; next
        Move_char_to_K);
      ¬ K_End  ∧ (WC ≠ 0) ∧ ¬ K_Data_Rq → (               idle till char arrives
        Move_char_to_K))
  Move_control_char_to_K := (                             K ← Pio ← M
     K_End → ;                                            stop at end
   ¬ K_End ∧ K_Data_Rq → (                                transmit a char
        (ASR = 0) → Move_word_from_M; next
        K_Data ← ARc[ASR]; ASR ← ASR + 1; next
        Move_control_char_to_K);
   ¬ K_End ∧ ¬ K_Data_Rq → Move_control_char_to_K))       idle, till char arrives
  Move_char_from_K := (                                   M ← Pio ← K data movement
      K_End ∨ (WC = 0)  → ;                               stop at end
      ¬ K_End ∧ (WC ≠ 0) ∧ K_Data_Rq → (                  receive a char
        ARc[ASR] ← K_Data; ASR ← ASR + 1: next
       (ASR = 0) → (Move_word_to_M; WC ← WC - 1); next
        Move_char_from_K);
      ¬ K_End  ∧ (WC ≠ 0)∧ ¬  K_Data_Rq → (               idle till char arrives
        Move_char_from_K))
  Move_word_to_M := (DR ← AR; next                        M ← Pio data movement
                     M[AC] ← DR; AC ← AC + 1)
  Move_word_from_M := (AR ← DR; next                      Pio ← M data movement
                       DR ← M[AC]; AC ← AC + 1)
  M_block_move := (                                       M ← M block move process for moving WC words within M, i.e.,
```

```
        (WC = 0) → ;                                    M[CC:(CC + WC)] ← M[AC:(AC + WC)]
        (WC ≠ 0) → (DR ← M[CC]; CC ← CC + 1; next
          M[AC]  ← DR; AC ← AC + 1; WC ← WC - 1; next
          M␣block␣move))
    K␣end␣wait := (                                     Process to idle until K transmits an end signal
      ¬ (K␣End ∨ K␣Unusual␣End) → K␣end␣wait;
        (K␣End ∨ K␣Unusual␣End) → ;)
```

APPENDIX 3 K('HYPERTAPE) AND 'KDISK ISP DESCRIPTIONS

Appendix 3

K('Hypertape) and K(disk) ISP Descriptions

These K depend on control and state definitions from Pio of Appendix 2.

K State

```
K␣op<0:1>16                                  the operation or instruction register in K
K␣Data<0:5>                                  data buffer in K; used for transmitting and receiving characters
K␣Data␣Rq                                    used to control data flow between ARc[ASR] and K␣Data: signal
                                               in K denoting K␣Data requires new data if writing, or has a
                                               full data buffer if reading
K␣End                                        set by K at the completion of reading or writing a block of data
K␣Unusual␣End                                set by K when an error is detected during writing or reading and
                                               data flow must be terminated
```

The following sense data bits for tape originate in Ms and K. These registers can be read by Pio using the Pio SNS instructions Some of the bits are set using the CTL, CTLR, or CTLW instructions from Pio as control words

```
SDT[0:1]<S,1:35>                             sense data for K('Hypertape)
   SDT[0]<1>/Operator Required := (
      SDT[0]<13>/Selected Drive Not Ready ∨
      SDT[0]<15>/Selected Drive Not Loaded ∨
      SDT[0]<16>/Selected Drive File Protected ∨
      SDT[0]<17>/Operation Not Started)
   SDT[0]<3>/Program Check := (
      SDT[0]<19>/Invalid Order Code ∨
      SDT[0]<21>/Selected Drive Busy ∨
      SDT[0]<22>/Selected Drive at Beginning of Tape ∨
      SDT[0]<23>/Selected Drive at End of Tape)
   SDT[0]<4>/Data Check := (
      SDT[0]<25>/Correction Occurred ∨
      SDT[0]<27>/Channel Parity Check ∨
      SDT[0]<28>/Code Check ∨
      SDT[0]<29>/Envelope Check ∨
      SDT[0]<31>/Overrun or Character Lost Check ∨
      SDT[0]<33>/Excessive Skew Check ∨
      SDT[0]<34>/Track Start Check or Clock Lost Check)
   SDT[0]<5>/Exception Conditions := (
      SDT[1]<1>/Selected Drive Read a Tape Mark ∨
      SDT[1]<3>/Selected Drive in End of Tape Warning Area)
   SDT[0]<7,9:11>/Selected Tape Unit Address 0:3
   SDT[1]<7>/Read Section Busy
   SDT[1]<9>/Write Section Busy
   SDT[1]<11>/Backward Mode
   SDT[1]<13,15:17,19,21:23,25,27>/Drive Attention[0:9]
SDF[0:1]<S,1:35>                             sense data for the K('Disk)
   SDF[0]<3>/Program Check := (
      SDF[0]<7>/Invalid Sequence ∨
      SDF[0]<9>/Invalid Code ∨
      SDF[0]<10>/Format Check ∨
      SDF[0]<11>/No Record Found
      SDF[0]<13>/Invalid Address)
   SDF[0]<4>/Data Check := (
      SDF[0]<15>/Response Check ∨
      SDF[0]<16>/Data Compare Check ∨
      SDF[0]<17>/Parity or Cyclic Code)
   SDF[0]<5>/Exception Condition := (
      SDF[0]<19>/Access Inoperative ∨
      SDF[0]<21>/Access Not Ready ∨
      SDF[0]<22>/Disk Circuit Check ∨
      SDF[0]<23>/File Circuit Check
   SDF[0]<7>/six Bit Mode/Status Bit
   SDF[0]<31,33:35>□SDF[1]<1,3:5,7,9>/Access 0, Module[0:9]
```

```
Control Orders, i.e
 Instruction Names and Numbers for K(disk)
 These instructions are set in the K op register by the CTL instructions from Pio.  The instructions are then executed by the
 K's.  They will only be given as names, mnemonics, and operation codes.
    DNOP (:= K␣op = AA) →                                   no operation
    DREL (:= K␣op = A4) →                                   release
    DEBM (:= K␣op = A8) →                                   eight bit mode
    DSBM (:= K␣op = A9) →                                   six bit mode
    DSEK (:= K␣op = 8A) →                                   seek
    DVSR (:= K␣op = 82) →                                   prepare to verify (single record)
    DWRF (:= K␣op = 83) →                                   prepare to write format
    DVTN (:= K␣op = 84) →                                   prepare to verify (track with no addresses)
    DVCY (:= K␣op = 85) →                                   prepare to verify (cylinder operation)
    DWRC (:= K␣op = 86) →                                   prepare to write check
    DSAI (:= K␣op = 87) →                                   set access inoperative
    DCTA (:= K␣op = 88) →                                   prepare to verify (track with addresses)
    DVHA (:= K␣op = 89) →                                   prepare to verify (home address)

Control Orders,.i.e.
 Instruction Names and Numbers for K('Hypertape)
    HNOP (:= K␣op = AA) →                                   no operation
    HEOS (:= K␣op = A1) →                                   end of sequence
    HRLF (:= K␣op = A2) →                                   reserved light off
    HRLN (:= K␣op = A3) →                                   reserved light on
    HCLN (:= K␣op = A5) →                                   check light on
    HSEL (:= K␣op = A6) →                                   select
    HSBR (:= K␣op = A7) →                                   select for backward reading
    HCCR (:= K␣op = 28) →                                   change cartridge and rewind
    HRWD (:= K␣op = 3A) →                                   rewind
    HRUN (:= K␣op = 31) →                                   rewind and unload cartridge
    HERG (:= K␣op = 32) →                                   erase long gap
    HWTM (:= K␣op = 33) →                                   write tape mark
    HBSR (:= K␣op = 34) →                                   backspace
    HBSF (:= K␣op = 35) →                                   backspace file
    HSKR (:= K␣op = 36) →                                   space
    HSKF (:= K␣op = 37) →                                   space file
    HCHC (:= K␣op = 38) →                                   change cartridge
    HUNL (:= K␣op = 39) →                                   unload cartridge
    HFPN (:= K␣op = 42) →                                   file protect on
```

APPENDIX 4 IBM 7094 PC INSTRUCTIONS TO PIO('7909)

Appendix 4

IBM 7094 Pc Instructions to Pio('7909)

Pc State

```
Pc␣trap␣enable<A,B,C,D,E,F,G,H>          An 8 bit register in Pc which is used to mask or allow trap
                                          requests from Pio.  (#A,B,...H)
```

Instruction Set
The following instructions in Pc are used to operate on each Pio state; thus, each instruction is actually 8 instructions.

```
RSC → (Wait → (CC ← e; Wait ← 0);          reset and start channel
      ¬Wait → RSC);                        initializes a Pio
STC → (Wait → (CC ← AC; Wait ← 0);         start the Pio program
      ¬Wait → STC);
SCH → (M[e]<21:35> ← CC; M[e]<3:17> ← AC); store channel.  Checks status of a Pio.
ENB → (Pc␣trap␣enable ← M[e]<28:35>);      enable from effective address
RIC → (CTC□AC□AR□CC□WC□Wait ← 0);          reset channel
TCO → (¬ Wait → IC ← e);                   transfer on channel in operation
TCN → (Wait → IC ← e);                     transfer on channel not in operation
```