IBM® Customer Engineering

Instruction-Reference

Preliminary Edition

IBM Confidential

7094 DATA PROCESSING SYSTEM

System Fundamentals
7110 Instruction Processing Unit
7109 Arithmetic Sequence Unit
7151-II Console Control Unit
7606 Multiplexor

R23-2550-1

~~R23-2550~~

1 2 3 4

Main Adders 205.1

Individual Adder 205.1

Lookahead Feature 205.1

Lookahead and Carry 205.2

Adder Carry 205.3

Group and Section Carry 205.4

Index Adders 208.1

Index Registers 209.1

Instruction Backup Register 210.1

Accumulator Register 210.2

Multiplier-Quotient Register 211.1

Address Register 211.2

Program Counter 212.1

PRogram Register 213.1

Sense Indicator Register 214.1

Shift Counter 215.1

Stepping the Shift Counter 215.2

7094 Double Latches 217.1

Read-In Sequence 217.3

Tag Register 219.1

Floating Point Tally Counter 220.1

1 2 3 4

7094 TIMING

Timing Function 300

Master Clock Pulses 301

Computer Set Pulses 307

Timing Cycles 310

Instruction Cycle 310

Execution Cycle 310

Logic Cycle 310

Buffer Cycle 311

Master I Time 312

Master E Time 320

Master L Time 322

Master B Time 323

7094 Lookahead Capability 324

!　+　#　$
1　2　3　4

7094 ARITHMETIC

- Binary Arithmetic 401.1
  - Addition 401.1
  - Subtraction 401.3
    - Complement Method 401.4
    - 1's Complement 401.6
    - 2's Complement 401.8
    - Signed Numbers 401.8
  - Multiplication 401.9
  - Devision 401.10
- Fixed Point Arithmetic 404.1
  - Addition 404.3
  - Subtraction 404.8
- Multiplication 404.13
  - 7094 Multiplication 404.44
  - Decoder Network 404.26
- Fixed Point Devision 404.34
- Variable Length Arithmetic 405.1
  - Variable Length Multiply 405.1
  - Variable Length Divide or Halt 405.3
  - Variable Length Divide or Proceed 405.3

1 2 3 4

Floating Point Arithmetic 406.1

Notation With the Binary System 406.2C

Floating Point Data 406.2D

Double Precision 406.2H

Floating Point Spill 406.2J

Normalizing 406.2K

Zero Fraction 406.2N

Arithmetic of Floating Point 406.3

Floating Point Arithmetic Timing 406.4

7094 Single Precision Floating Point Add 406.13

E Time 406.13

~~FACT 1~~

Floating Add Control Triggers 406.14

FACT 1 406.14

FACT 2 406.14

FACT3 406.15

FACT 4 406.15

FACT 5 406.15

FACT 6 406.16

FACT 7 406.17

FAD End Op Conditions 406.24

Floating Point Multiply 406.30

E Time 406.33

L Time 406.34

1 2 3 4

7094 Double Precision Floating Point Add 408.0.1

DPS 0 408.0.1

DPS 1 408.0.1

DPS 2 408.0.2

DPS 3 408.0.2

Double Precision Floating Point Multiply 408.30

7094 DPFM 408.35

E Time of DPFM 408.36

Second Multiply 408.38

Third Multiply 408.39

Double Precision Floating Point Devide (DPFD) 409.1

Fir st Step L Time 409.3

L Time Second Step 409.6

L Time Third Step 409.7

1 2 3 4

INSTRUCTIONS

Word Transmission Instructions 500.1

Store 500.1

Store Logical Word 500.1

Store MQ 500.1

Store Zero 500.1

Store Prefix 500.1

Store Decrement 500.3

Store Tag 500.3

Store Address 500.3

Store Left Half MQ 500.3

Store Instruction Location Counter 500.3

Load MQ 500.3

Exchange AC and MQ 500.3

Exchange Logical AC and MQ 500.3

Enter Keys 500.6

Shifting Instructions

Accumulator Left Shift 500.6

Long Left Shift 500.6

Logical Left Shift 500.6

Accumulator ~~Left Shift~~ Right Shift 500.8

Long Right Shift 500.8

Logical Right Shift 500.8

Rotate MQ 500.8

1 2 3 4

Transfer Instructions

Transfer	500. 11
Transfer on MQ Plus	500. 11
Transfer om Plus	500. 11
Transfer on Minus	500. 11
Transfer on Overflow	500. 11
Transfer on No Overflow	500. 11
Transfer on Quotient Overflow	500. 11
Transfer on Zero	500. 11
Transfer on No Zero	500. 13
Transfer on Low MQ	500. 13
Transfer on Channel in Operation	500. 13
Transfer on Channel not in Operation	500. 13
Transfer on Data Channel Redundancy Check	500. 16
Transfer on Data Channel End of File	500. 16

Skip Instructions

P Bit Test	500. 17
Low-Order Bit Test	500. 17
Storage Zero Test	500. 17
Storage Non-Zero Test	500. 17
Compare Accumulator with Storage	500. 17 [illegible]
Logical Compare Accumulator with Storage	500. [illegible] 20
Plus Sense	500. 20

1 2 3 4

Minus Sense 500.22

Input-Output Check Test 500.22

Divide Check Test 500.22

Beginning of Tape Test 500.25

End of Tape Test 500.25

Control Instructions

Halt and Proceed 500.25

Halt and Transfer 500.25

No Operation 500.25

Execute 500.25

Set Sign Plus 500.25

Set Sign Minus 500.29

Change Sign 500.29

Sense Indicator Instructions

Load Indicators 500.30

Store Indicators 500.30

OR Stprage to Indicators 500.30

Invert Indicators from Storage 500.30

Reset Indicators from Storage 500.30

Set Indicators of Right Half 500.30

Set Indicators of Left Half 500.33

Reset Indicators of Right Half 500.33

1 2 3 4

~~Re~~

Reset Indicators of Left Half 500.33

Invert Indicators of ~~Left Half~~ Right Half 500.33

Invert Indicators of Left Half Half 500.33

Place Indicators in Accumulator 500.33

Place Accumulator In Indicators 500.33

OR Accumulator to Indicators 500.35

~~Reset Accumulator from~~
Reset Indicators from Accumulator 500.35

Invert Indicators from Accumulator 500.35

Transfer if Indicators On 500.35

Transfer if Indicators Off 500.35

On Test for Indicators 500.35

Off Test for Indicators 500.38

Righ Half Indicators, On Test 500.38

Left Hald Indicators, On Test 500.38

Right Half Indicators, Off Test 500.38

Left Half Indicators, Off Test 500.38

Index Instructions 500.40

Load Index from Address 500.42

Load Index from Decrement 500.43

Load Index From Address Complemented 500.44

Load Index from Decrement Complemented 500.45

1 2 3 4

Address to Index True 500.47

Address to Index Complemented 500.48

Place Address in Index 500.50.1

Place Decrement in Index 500.50.2

Place Address in Index Complemented 500.50.3

Place Decrement in Index Complemented 500.50.4

Store Index in Address 500.50.6

Store Index in Decrement 500.50.8

Store Complement of Index in Address 500.50.9

Store Complement of Index in Decrement 500.50.10

Place Index in Address 500.51.0

Place Index in Decrement 500.51.2

Place Complement of Index in Address 500.51.3

Place Complement of Index in Decrement 500.51.4

Transfer on Index 500.52

Transfer on No Index 500.54

Transfer on Index High 500.55

Transfer on Index Low 500.56

Transfer With Index Incremented 500.58

Transfer and Set Index 500.61

AND and OR Instructions

OR to Storage 500.63

OR to Accumulator 500.63

AND to Accumulator 500.63

AND to Storage 500.63

Exclusive OR to Accumulator 500.63

Convert Instructions 500.68

Convert by Replacement from Accumulator 500.69

Convert By Replacement from MQ 500.71

Convert by Addition from MQ 500.73

1 2 3 4

7094 TRAPPING

Trap ModeInstructions 601.1

Enter Trapping Mode 601.1

TRA in Trapping Mode 601.1

Leave Trapping Mode 601.1

Trap Transfer 601.1

Store Location and Trap 601.1

1 2 3 4

INSTRUCTION OVERLAP 700

Data Lookahead 707

Transfer Lookahead 715

Store Lookahead 735

1 2 3 4

IBM 7151 CONSOLE CONTROL UNIT 800

Operator's Panel 802

Indicators 802

Internal Registers 802

Trap 802

Simulate 802

Accumulator Overflow 802

Quotient Overflow 802

Read-Write Select 802

Divide Check 803

Channel Selsect (A-H) 803

Command Word Trap (Chan A-H) 803

Tape Check Trap (Chan A-H) 803

Channel Tape Check 803

Trap Control 803

Program Stop 803

I-O Check 803

Ready Light 804

Automatic 804

Console Power-On 804

Central Components Power Check 804

I-O Power Check 804

Power 804

Marginal Check +6 804

1 2 3 4

Marginal Check -12v	804
Manual Controls	
Power On	805
Normal-Off	805
Emergency Off	805
Reset	805
Automatic-Manual Key	805
Entry Keys	807
Manual Control Keys	807
Start Key	807
Clear Key	807
Didplay Storage	807
Display Indicators	807
Display Effective Address	811
Single Step	811
Multiple Step	811
Enter MQ Key	811
Enter Instruction	811
Load Cards and Load Tape Keys	811A
Sense Control	814

1 2 3 4

Customer Engineer's Test Panel

Indicators 814

Address Register 814

Cycle Time 81~~4~~6

Tally Counter 81~~4~~6

T-2 81~~4~~6

FP 81~~4~~6

9 Carry 81~~4~~6

9 Overflow 81~~4~~6

Q Carry 81~~4~~6

Master Stop 816

End Operation 818

AND 818

CAQ 818

X Carry 818

Switches

I-O Interlock Switch 818

Continuous Enter Instruction 818

Multiple Step High and Low Speed 819

B Cycle Controls 819

Machine Cycle Key 819

Suppress TLA 821

Suppress DLA/SLA 821

Display IBR 821

1 2 3 4

Memory Diagnostic Mode 821

Machine Cycle Jack 821

Auxiliary Start and Reset Jack 821

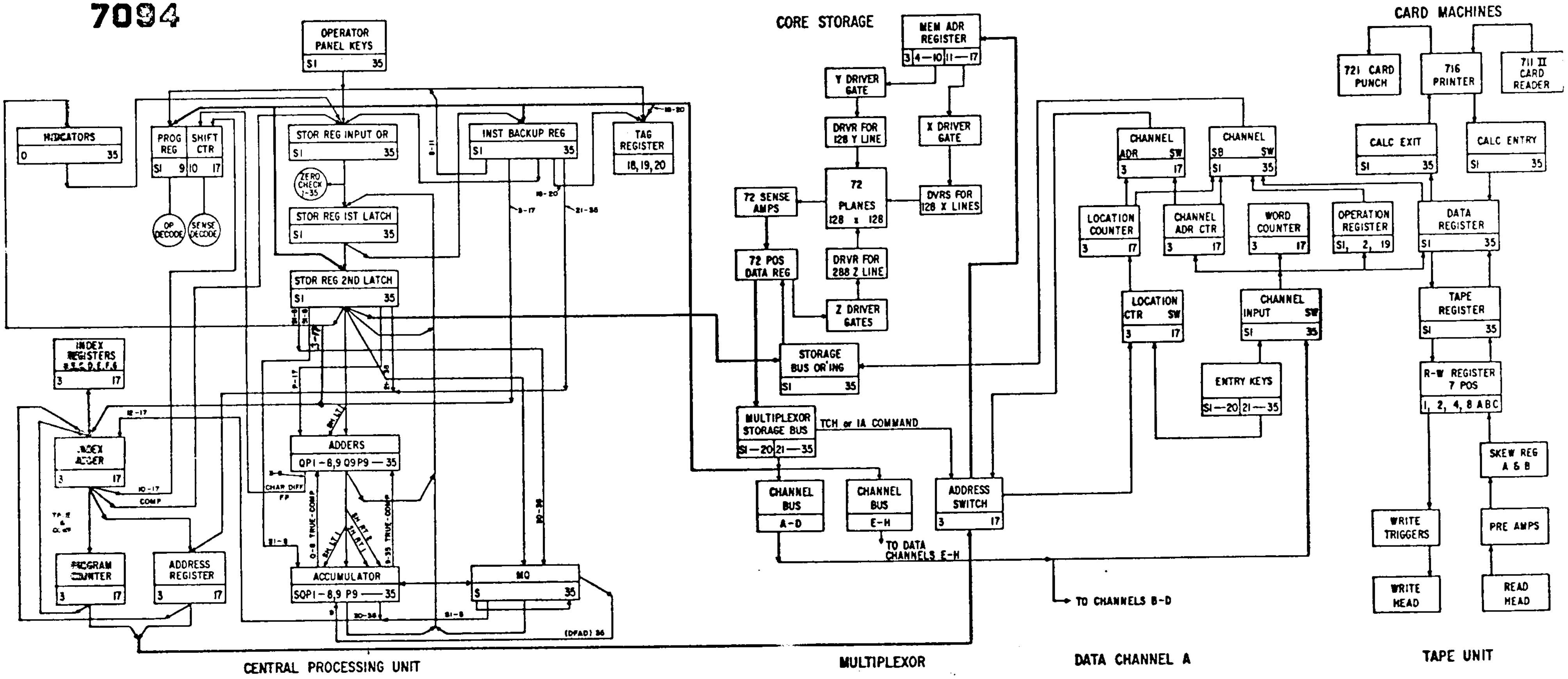
Phone Jack 822

16K/24K Mode Switch 822

DC On 822

Marginal Check 822

7094


CORE STORAGE
CARD MACHINES
OPERATOR PANEL KEYS SI 35
MEM ADR REGISTER 3 4—10 11—17
Y DRIVER GATE
X DRIVER GATE
DRVR FOR 128 Y LINE
DVRS FOR 128 X LINES
72 PLANES 128 x 128
DRVR FOR 288 Z LINE
Z DRIVER GATES
72 SENSE AMPS
72 POS DATA REG
STORAGE BUS OR'ING SI 35
MULTIPLEXOR STORAGE BUS SI—20 21—35
TCH or IA COMMAND
CHANNEL BUS A–D
CHANNEL BUS E–H
TO DATA CHANNELS E–H
ADDRESS SWITCH 3 17
TO CHANNELS B–D
721 CARD PUNCH
716 PRINTER
711 II CARD READER
CALC EXIT SI 35
CALC ENTRY SI 35
CHANNEL ADR SW 3 17
CHANNEL SB SW SI 35
LOCATION COUNTER 3 17
CHANNEL ADR CTR 3 17
WORD COUNTER 3 17
OPERATION REGISTER SI, 2, 19
DATA REGISTER SI 35
LOCATION CTR SW 3 17
CHANNEL INPUT SW SI 35
ENTRY KEYS SI—20 21—35
TAPE REGISTER SI 35
R-W REGISTER 7 POS 1, 2, 4, 8 ABC
SKEW REG A & B
WRITE TRIGGERS
PRE AMPS
WRITE HEAD
READ HEAD
INDICATORS 0 35
PROG REG SI 9
SHIFT CTR 10 17
OP DECODE
SENSE DECODE
STOR REG INPUT OR SI 35
ZERO CHECK 1-35
STOR REG 1ST LATCH SI 35
STOR REG 2ND LATCH SI 35
INST BACKUP REG SI 35
TAG REGISTER 18, 19, 20
18-20
S-11
3-17
18-20
21-35
INDEX REGISTERS A B C D E F G 3 17
INDEX ADDER 3 17
12-17
10-17
COMP
TP II & OUT
PROGRAM COUNTER 3 17
ADDRESS REGISTER 3 17
ADDERS QP1—8,9 Q9 P9 — 35
S-8
CHAR DIFF FP
0-8 TRUE-COMP
9-35 TRUE-COMP
SH LT 1
SH RT 1
SH RT 8
SH LT 1
S1-8
ACCUMULATOR SQP1—8,9 P9 — 35
30-35
MQ S 35
(DFAD) 35
S1-8
P-17
3-17
S-8
S1-8
CENTRAL PROCESSING UNIT
MULTIPLEXOR
DATA CHANNEL A
TAPE UNIT

IBM 7094 DATA PROCESSING SYSTEM

FUNDAMENTAL PRINCIPLES

Modern methods of accounting, measuring, testing, research, and design generate huge quantities of information that must be processed quickly and accurately. A vast amount of data constantly pours into such places as retail establishments, weather stations, insurance companies, and tax bureaus. In addition, our rapidly expanding scientific investigations into rocket and missile design, atomic research and missile tracking need faster and faster methods for carrying out increasingly complex calculations. To meet these demands, machines have been developed which can compute, select, and correlate data at electronic speeds.

Automating paper work is possible because the actions involved are sufficiently repetitive. The variety of steps necessary in processing business records or in computing scientific problems, for example, is small, indeed, compared with the number of times these steps must be taken. One of the first paper work machines was the ink stamp, possibly because applying a data or name was so obviously repetitive, As additional mechanization was applied to paper work, machines begin to take over the long and pains-taking task of accounting.

Although accounting applications of business machines require a certain amount of arithmetic, such as accumulating totals and balances, the problem is principally one of processing data. A large amount of information is fed into these machines (input) and a relatively small amount of information is produced (output). The machines are therefore called data processing machines.

The first data processing machines handled information in a series of individual operations. These included punching information into cards, sorting and classifying cards, producing totals and balances, and printing the results. Intermediate results from one machine were transferred to another, many human decisions and interventions were necessary for a complete accounting procedure.

With the application of electronics, the rate of calculation was vastly increased. But, more important, a basic new technique was introduced which might be called intercommunication. Electronic devices were able to make decisions, and, on the basis of these decisions, to provide internal transporting of data and intermediate results from step to step. Data are fed into one end of a data processing system and final results come out the other, This machine system, as we know it today, is the modern computer, or data processing system.

GENERAL COMPUTER FORMAT

A modern computer has five distinct functional sections: input, storage, arithmetic and logical, control and output, These same general functions exist, to some degree, in all IBM card machines. Figure 100 illustrates these functions as they appear in an IBM card machine accounting system.

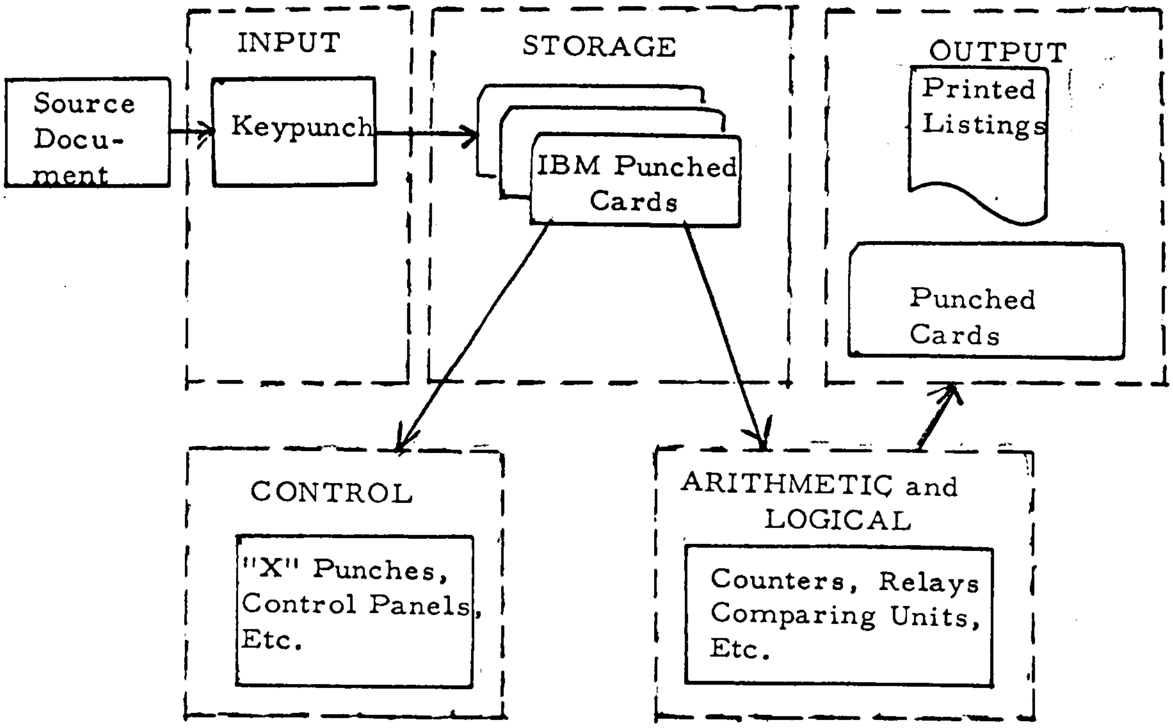


Figure 100 Functional Parts of a Card Accounting System

All information that is to be involved in any mechanical method of accounting must be made available in a language and format that the machines can understand. This function is provided in the card accounting system by the keypunch. The printing on source documents such as billing or various account statements is converted to code holes in the punched card. The keypunch is, then, the input to the card accounting system. Once the card is punched, it serves as a storage device. The information stored on the card is now available for future use by all portions of the accounting system. Exactly how this stored information is to be used in the accounting system is controlled by several factors. These factors include such items as control punches in the cards, the order in which the cards are used, the machines in which they are used, and the wiring of the control panels on the machines. The summation of these factors represents the control sectionof the card accounting system. Items such as counters, comparing units, selector units, and certain relay circuits constitute the arithmetic and logical section of a card accounting system. Which of these units operates on the information from the cards, and in what manner is dictated by the control portion of the accounting system. The results or output obtained from the accounting system are usually delivered as printed listings. If the results are to be used, the output is also punched into cards.

It is important to realize that the machines which comprise a card accounting system must receive detailed directions from the control portion of the system. The control section in turn, is directed by control panel wiring, control punches in the cards, and the card routing to the particular machines. It will be seen that a computer must receive even more specific directions than those used in a card accounting system because a computer does absolutely nothing unless directed to do so by its control section. The control section of the computer receives its directions from a programmer who plans the operations to be performed by the computer.

The five functional sections of a generalized computer are illustrated in Figure ~~2~~101. In general, the computer is similar to an entire card accounting system. The advantages to be realized by using a computer include greatly increased processing speeds, a higher degree of automation, and greater flexibility.

All information used by the computer must pass through the input section where the incoming information is interpreted and converted to the language that the computer understands. The input section includes such devices as card readers and magnetic tape units.

From the input section, information is directed to the storage section. As their main storage unit, most computers use an information-holding device composed of magnetic cores. This magnetic core storage may serve as the source of all information to be used by the computer. Core storage has some very important advantages over punched card storage. Most important of these is the high speed at which information may be placed in, or removed from, core storage. The highest degree of performance from core storage and many other types of storage can be realized only when the information is arranged in specific order. Once the information is located in core storage, it may be called for instantly in any sequence.

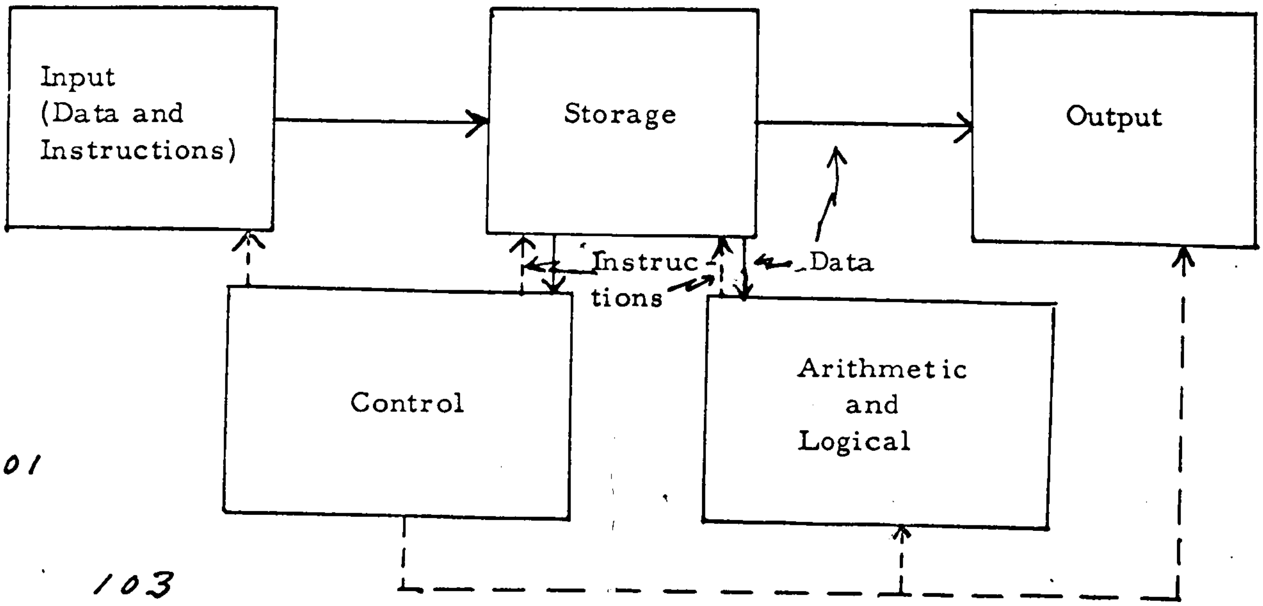


Fig. 101

The control section of a computer directs the operation of the entire computer. Unlike the card system, the control section of a computer receives its directions as units of detailed information from core storage. These units of information, which tell the control section what operations are to be performed are called instructions. That portion of the information in storage which is to be operated on is commonly referred to as data. A single piece of data used in an operation is often called an operand. From the above, it may be seen that instructions as well as data must be delivered to storage from the input section. Notice in Figure 2 that the control section receives instructions from storage and then exhibits the necessary control over all other sections of the computer.

The actual operations are, for the most part, performed in the arithmetic and logical section. The control section directs exactly what operation is to be performed and what operand is to be involved in the operation. The instructions that may be executed by a given computer includes such arithmetic operations as add, subtract, multiply or divide, Some instructions may place the results of the arithmetic operation back in core storage. Subsequent instructions may tell the control section to deliver the information to the output section. The output section may include printers, punches, or magnetic tape units.

From the above description, it may be seen that a single instruction causes only a specific operation to be performed by the computer. If a complete problem is to be performed by the computer, a number of instructions are required to direct the computer. A group of such instructions, and any necessary constant data used to direct the computer to the accomplishment of a job, is called a program. Because, in modern computers the program is contained in storage, the computers are called "stored program computers." When operating under stored program control (or any other method of control), the computer executes one instruction at a time. After executing one instruction, the computer automatically proceeds to the next instruction. It is important to realize that every computer must be directed by some type of program during every step of its operation.

A generalized stored program computer, as illustrated in Figure ~~2~~ 101, operates in the following manner. A program of instructions to direct the computer in every step of its operation is punched into IBM cards. All of the data upon which the computer is to operate are also punched into cards, The cards are placed in the card reader, and a key on the computer console is pressed which tells the control section that the information located in the card reader is to be read into storage. The control section then starts the card reader and delivers the information from the card reader to the proper locations in storage. Information contained in the last card tells the control section where to find the first instruction. The control section then calls for and decodes the instruction to determine what operation is to be performed and where the operand that is to participate is located. Next, the control section causes the operand-also located in storage-to be delivered to the arithmetic and logical section. The arithmetic section then performs the operation as called for by the control section. After the first instruction has been performed, the control section calls for the next instruction from storage. This instruction may very well be one that causes the results of the previous instruction to be stored. This process continues until such time that an instruction is encountered that causes the results (now located in storage) to be delivered to the output section.

THE 7094 FORMAT

Figure ~~2~~ 103 illustrates the organization of the 7094. Although there is a slight change in terminology, components and functions are essentially the same as previously described for a general computer. The central porcessing unit (CPU) of the 7094 is actually made up of two sub-units -- CPU 1 and CPU 2. CPU 1 is the IBM 7110 Instruction Processing Unit and CPU 2 is the IBM 7109 Arithmetic Sequence Unit.

As these names imply, CPU 1 contains all arithmetic and control registers and accepts, decodes, and routes instructions to the rest of the computer. CPU 2 controls instruction execution controls for all instructions except complex arithmetic and I-O controls ~~contains registers to perform the arithmetic functions~~. There is a great deal of interplay and overlap between these two units, and in some sequences it is difficult to determine an accurate functional boundary.

Note that a multiplexor has been added in Figure 3. The multiplexor processes most of the intercommunication within the system. Some kind of multiplexing is required since it is possible for the CPU to be in an arithmetic sequence, not requiring reference to core storage, at which time information may still be brought in from the input section. As will be explained in detail later, such simultaneous operation is highly conditional.

Priefly, computer information flow in Figure 3 is from input through multiplexor to CPU, to set up the stored program. Instructions then come from core storage, through the multiplexor to CPU for decoding and reference is made back to core storage to the address specified in the instruction. Finally, core storage instructions are necessary, to place the answers in core storage to be read out to the output equipment.

Figure 4 is a block diagram of the 7094 showing a possible combination of units making up a 7094 system. Of course, not all units are required in every installation. The final selection would be determined by customer need.

THE 7094 COMPUTER WORD

Before a computer can be told what to do, a common language is necessary between programmer and computer. The 7094 is a binary machine, so all inputs and outputs, internal processing, and internal communication is in terms of 1-bits 0-bits. These 1's and 0's are combined in a 36-bit word, in such combinations that are meaningful to the computer.

For example, the combination of 000100000, properly placed in the computer word, instructs the computer to perform an addition. Another portion of this particualr 36-bit word tells the computer which word in core storage is to be added (the operand). The next instruction might contain the combination of 000110000, which, in the proper location within the word, instructs the computer to store the sum just obtained. Again, this address portion of this word will instruct the computer where the sum is to be placed in core storage.

From the foregoing, two types of words are apparent -- instruction words and data words. A third type exists, the channel command. Channel commands tell the computer which input or output device is to be used for a particular operation, such as read or write. These three types of words -- data, instruction and command -- are arranged by the programmer into a logical sequence that will result in problem-solving or data processing to a desired end result.

With three kinds of words, made up only of 1's and 0's, it becomes apparent that the computer must be made to distinguish between them in order to perform the proper operation. Some reference must be established against which the computer can compare the word to tell immediately what action it is to take. This reference is established by timing.

A word occurring at a certain time is recognized as an instruction or an operand or a channel command. Timing is established by a 3-megacycle oscillator in the multiplexor. ~~— the computer clock.~~ Output pulses from this clock are shaped, inverted, and gated to become clock and set pulses. Twelve clock pulses constitute a computer cycle, having a total duration of 2 microseconds. Each pulse, therefore, is ~~166~~ 167 nanoseconds long.

These groups of 12 clock pulses are gated by appropriate circuits to become instruction, execution, logic, or ~~B~~ buffer cycles. Now, a word coming into the CPU at the beginning of an instruction, or I cycle, is immediately identified as an instruction. The execution, or E, cycle instructs the computer to refer to core storage for an operand. A logic, or L, cycle means no reference to core storage is necessary but that arithmetic or logic operations are to be performed. B time always means the use of input or output equipment.

Timing in the 7094 is critical, and for this reason a section of this manual has been devoted to timing sequences, gating, and the development of permissive and non-permissive circuits which allow the computer to function properly. A thorough understanding of computer timing is essential to understanding operation of the 7094.

COMPUTER SECTIONS

The input section of a computer system accepts information from any outside source and places it in the storage section. This information may come from punched cards, magnetic tape, or manually operated keys. The information may be instructions data (numbers for arithmetic calculations), or alphabetic characters for printing page headings, comments, and so forth.

The storage unit accepts and stores information that comes into the system through the input section. When any portion of the information in storage is needed, that portion is located and sent out to the section requesting it. All information in the system is at one time or another, in storage; therefore, computer speed depends on storage speed. The storage scheme of most computer systems today is random access -- any portion of information can be ~~lock~~ located directly without searching other locations.

The arithmetic section is the calculating section of the computer system. Here, portions of information, either instructions or data, canbe transformed, combined, or altered.

The control section directs the other sections. It tells them what to do and when to do it. Instructions come into the control section from storage. The control ~~sec~~ section also controls itself in that it keeps account of the instruction it is using and the one that it will use next.

The output section takes calculated information from storage and presents it to an outside user. Commonly used forms of outputs are : information on magnetic tape, punched cards, printed reports or indicator lights.

7094 SYSTEM MAKE-UP

The 7094 system includes all five of the sections previously mentioned. Figure 104 shows the general grouping of these sections in the 7094 system. Arrows indicate the general flow of information. Although the sections can be neatly separated physically, there are many functional combinations not shown in this grouping. Input and output are combined with a portion of control in a data channel, and arithmetic with another portion of control

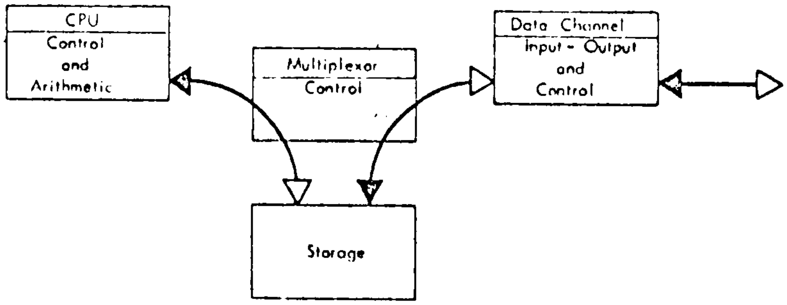


Figure 103. 7094 System Functional Arrangement

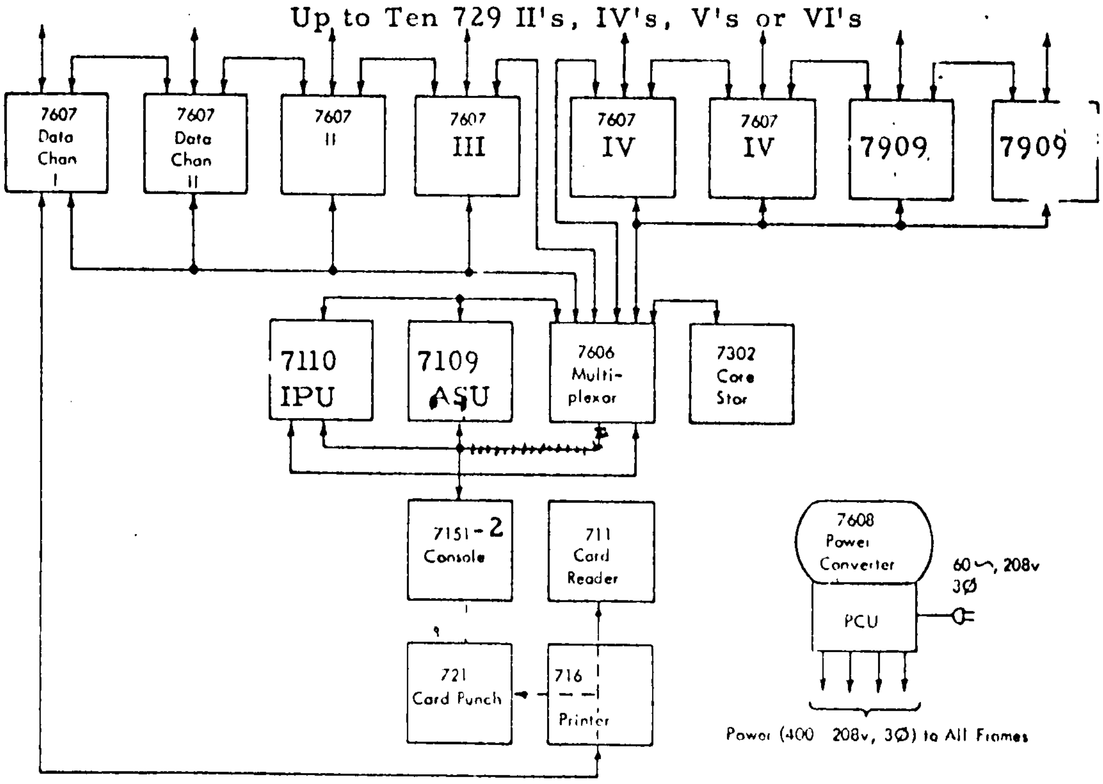


Figure 104. Block Representation of 7094 System

in the computer. Storage is the only functional section that is a separate machine unit. The multiplexor controls routing information into and out of storage.

The arrangement shown allows input-output (I/O) to operate somewhat independently, sharing storage with the computer. The highest order of ~~xo~~ controls is in the computer, where control ~~ixxxk~~ is delegated to the lower order controls in the data channel and multiplexor.

Figure 10~~4~~5 is representative of a 7094 system, showing the grouping of 7094 functions with machine types. The 7110 Inst5uction Processing Unit and the 7109 Arithmetic Sequence Unit are jointly referred to as the computer.

The number of machines applicable to 7094 input-output (I/O) operations is variable and includes the following :

7607 Data Channel Models 1/3	7607 Data Channel Models 2/4	7909 Data Channel
716 Printer	729II Magnetic Tape Unit	7631 Disk Storage
721 Card Punch	729 IV Magnetic Tape Unit	7640 Hypertape
711 Card Reader	729 V Magnetic Tape Unit	1414-6 I/O Synchronizer
729 Tape Units	729 VI Magnetic Tape Unit	7750 Programmed Transmission Control

The reader, punch and printer can only be used with the 7607 Data Channel models 1 and 3. The 729 II and 729 IV tape units can be intermixed on any model data channel. The 729 V and 729 VI tape units can only be used on the 7607 Data Channel models 3 and 4.

The computer is the control center of the 7094 Data Processing System. The computer also performs all arithmetic and logic functions. The computer receives information fr om storage, decodes it and performs the necesaary operations. Even thoug I/O is an independently functioning section, its operation must be initiate by the computer.

The IBM Magnetic Tape Units 'rite information on magnetic tape or

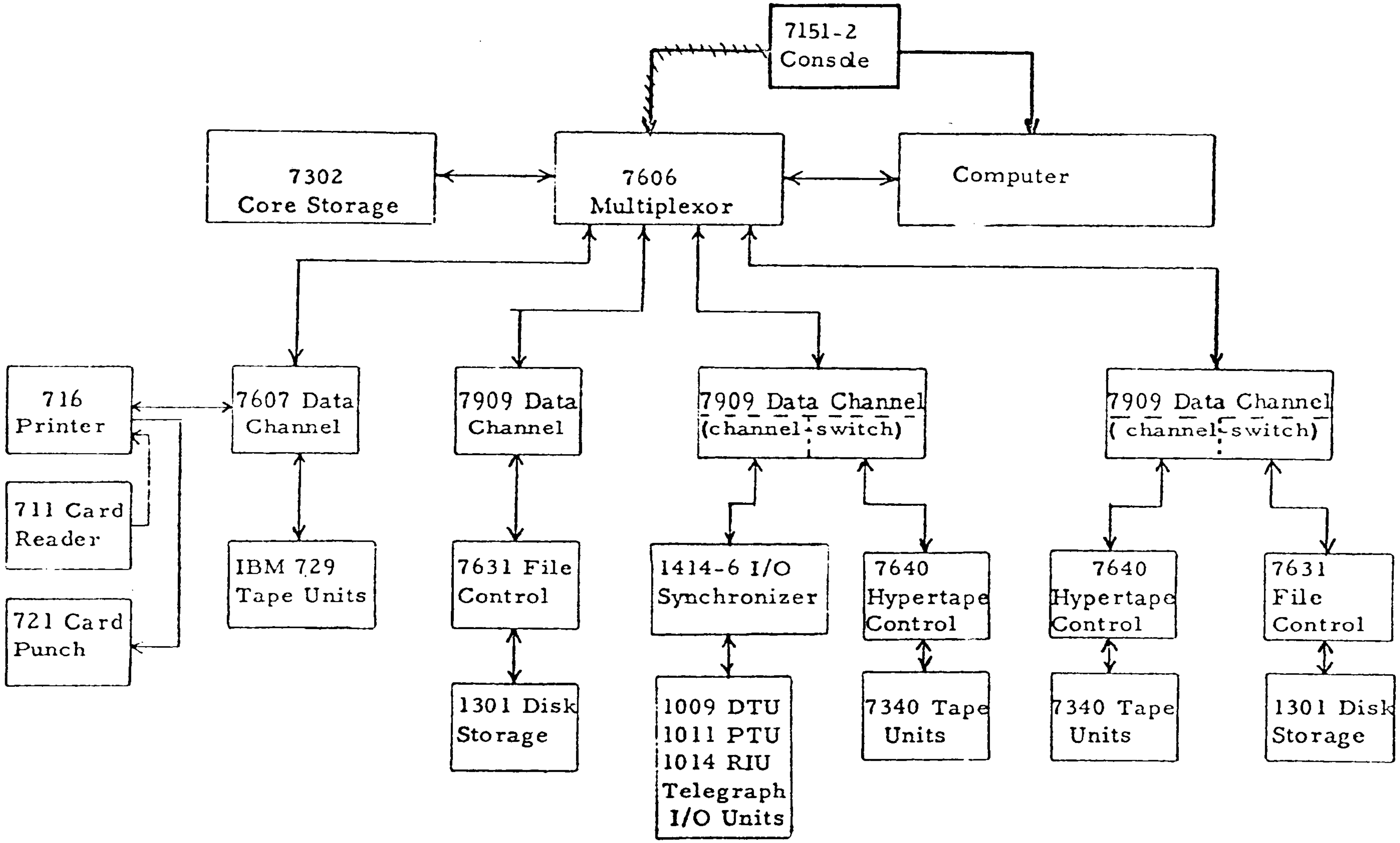


Figure 105 . IBM 7094 Data Processing System Configuration read information from magnetic tape. The higher model numbers indicate advanced versions of the basic unit -- usually referring to an increased character rate capability.

The IBM 711 Card Reader reads information from punched cards at 250 cards per minute.

The IBM 716 Printer prints information from core storage at 150 lines per minute. The typewheel echo pulses are available to the 7094 system, where they may or may not be used to check the accuracy of printing.

The IBM 721 Card Punch punches information information from core storage at 100 cards per minute.

The IBM 7607 Data Channels control the flow of information between the I/O units and core storage. The model 1 can control any combination of ten 729 II and 729 IV tape units and up to one of each reader, punch and printer. The printer must be present if either a reader or a punch are to be used. A 7607 Model II can control ten tape units, but neither card machines nor printer. The 7607 Models 3 and 4 perform the same functions as models 1 and 2 but with the added capacity for handling 729 V and VI tape units. The 7094 Data Processing System may include up to eight data channels.

Each data channel can be regarded as a subsystem, with its own manual control console and indicator panel (not shown on the drawings). Once a data channel is set ;in operation by an instruction in the computer program, it can call its own instructions (called commands in channel operations). These commands make up what is known as an I/O program. This program controls the operation of the I/O unit. Information received from an I/O unit is placed in core storage, or information is taken from core storage to be supplied to a

The IBM 1414-6 Input - Output Synchronizer permits the attachment of communications and paper tape devices such as :

IBM 1009 Data Transmission Unit

IBM 1011 Paper Tape Reader

IBM 1014 Remote Inquiry Units

Telegraphic Input-Output Units

The IBM 7302 Core Storage is a fast, random-access storage unit. A unit of information can be read into or out of any of its 32,768 storage locations in two microseconds. Storage serves both the computer and the data channels. The only restriction is that no two units can be using storage at exactly the same time. If a data channel calls for storage while the computer is using storage, the channel waits until the computer permits storage priority to pass to the channel.

The IBM 7606 Multiplexor is a time-sharing and switching device. It provides a path to and from storage for the computer and the data channels. The multiplexor also performs certain anticipatory, or lookahead functions associated with data channel operations.

The IBM 7151-2 Console Control Unit provides a manual means of controlling the system and displaying, by indicator lights, the contents of various registers, or any one of the storage locations. Several registers are continually displayed. The console also contains a CE test panel and a marginal voltage check panel.

The IBM 7750 Programmed Transmission Control links a central computer with telecommunication equipment such as telegraph machines, IBM 65/66 Data Transceivers, and IBM 7701 Magnetic Tape Transmission Terminal.

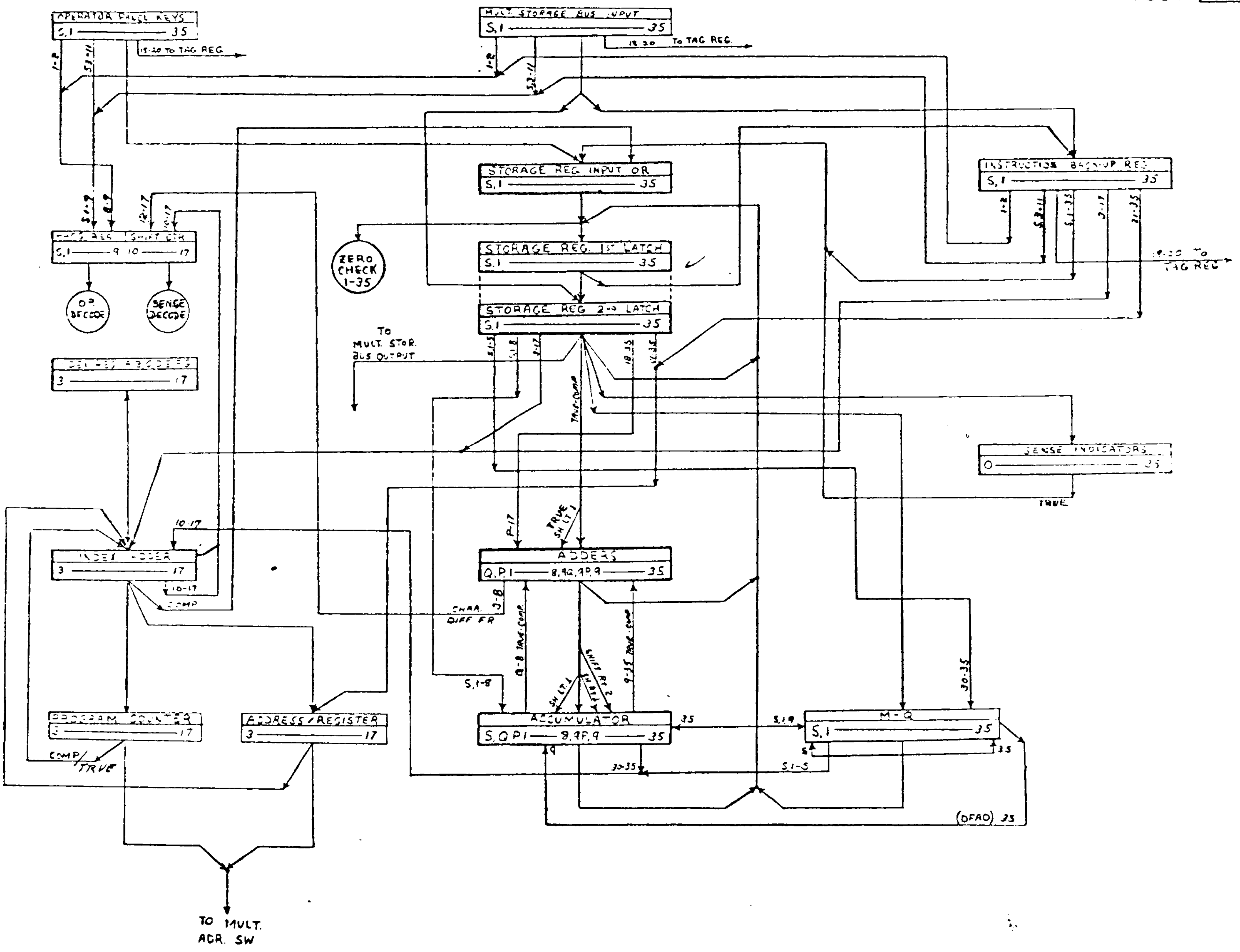


Figure 2 7094 Flow Diagram

7094 FUNCTIONAL COMPONENTS

2#

7094 COMPUTER WORD

A 7094 word may be a numeric quantity, an instruction to the computer or a data channel command. In all cases, the word contains a full 36 positions or bits. The contents of the word become significant according to the cycle of operation the computer is in. Thus, a word coming into the computer during an instruction cycle is treated as an instruction. Also, a word coming into the computer during an execution cycle is treated as a numeric quantity.

The computer is incapable of distinguishing kinds of words by their content alone but it will be shown that bit positions vary in significance according to the computer cycle.

Data Word

When the 36 bits are expressing a numeric quantity the word is referred to as a data word. Figure 201-1 shows a data word. Notice the numerical value is expressed in positions 1 through 35 and the sign of the value is expressed in the 0 or S position. When S is a 0, the value is positive. When S is a 1, the value is negative.

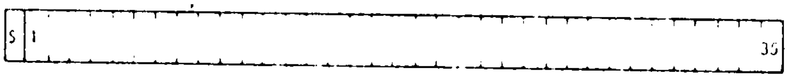


Figure 201-1 7094 Data Word

Instruction Word

A computer instruction word is shown in Figure 201-2. Because this word is coming into the computer during an instruction cycle it will, in effect, be segmented, and the various segments interpreted to determine what action is expected of the computer. The 36 bits of the computer word are now broken into significant sections -- prefix, decrement, tag and address.

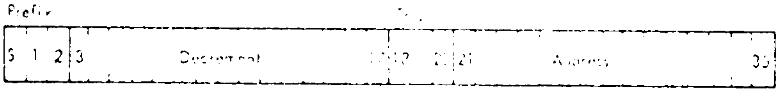


Figure 201-2.ʼ7094 Instruction Word

The sign position is always a part of the operation code. Along with the sign, either the remainder of the prefix field, or the decrement dictates the function to be performed. If positions 1 and 2 contain zeros, the sign and decrement determine the operation code. If either or both positi ons 1 and 2 contains a 1, the prefix contains the entire operation code and the decrement is used for another purpose.

The address field usually contains the address of a data word in core storage. This data word is brought into the computer as a part of whatever arithmetic or logic function is called for by the operation code. Thus, the instruction not only dictates the operation to be performed, but also specifies the address of the data to be used. In some instances, the address field is a part of the operation code. When this is the case, the address field is not used to address the data in storage.

The tag field causes the computer to calculate a storage address different from the address field of the instruction.

Following are variations of the computer instruction word.

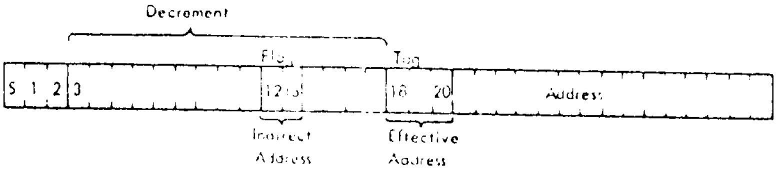


Figure 201-3. Instructio: Word address Modification Fields

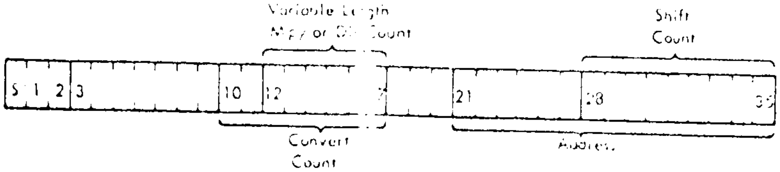


Figure 201-4. Instruction Nord Count Fields

Data Channel Command Word

Similar in format and application to the computer instruction word, the data channel command word , Figure 201-5, gains its special significance by being called out of storage by the control function in the data channel. This, as in the computer, occurs when one operation has been completed and the data channel must be directed to perform the next operation. The two major differences are :

1. The prefix is always the operation code in the data channel.
2. Positions 3 - 17 are a word counter. Whether the operation is reading or writing, once the command is in the channel, the word count must become one less each time a word is handled. Thus, when the count becomes zero, the channel must ask for a new command.

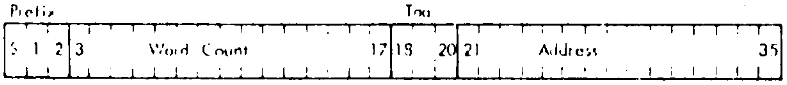


Figure 201-5. Data Channel Command Word

IBM 7302 CORE STORAGE (2.0-usec)

Some of the 7302 Core Storage Units (2.0-usec) have been modified to accommodate the 7094 Data Processing System. As a result, the storage cycle time of the modified machine has been reduced to 2.0-usec. In addition, it is possible to gate out two words (72 bits) to the computer on one storage cycle. Thus, full advantage is taken of the 72-bit readout feature of the 7302. Because these innovations represent comparatively few circuit changes, this section will cover only the affected circuits. It may be assumed that the descriptions of the remaining circuits, as presented in the previous sections, are applicable to the modified machine.

WORD PATTERN (7094)

The word pattern used in 7094 operations differs from that used by the 7090 system. In the 7090, the lower word (planes 0-35) and the upper word (planes 36-71) are a mirror image of each other, i.e., bit positions 0 and 71 represent sign, and bits 35 and 36 are the 35th (last) position of each word. In the 7094, however, the sign positions are 0 and 36, and bits 35 and 71 are the last position of each word. This can be determined by Figure 94, which shows the MDBI to MDR cabling for the modified machine. The typical cabling circuit (Figure 94A) gives all the connections between the MDBI, and MDR positions 35 and 71. Because they are jumpered together, the status of positions 35 and 71 are the same in their respective words. Compare this circuit with the typical MDBI to MDR cabling circuit for the 2.18-usec machine (Figure 62). Figure 94B gives the jumpering arrangement for all positions.

STORAGE LOCATIONS

Recall that when an address was presented to the 2.18-usec machine, the two words subsequently read out of storage consisted of the requested word, and the nonselected word located in the same relative position in the opposite half of memory, but displaced by 40000_8. For example, if the word in location 00176_8 was requested, the word

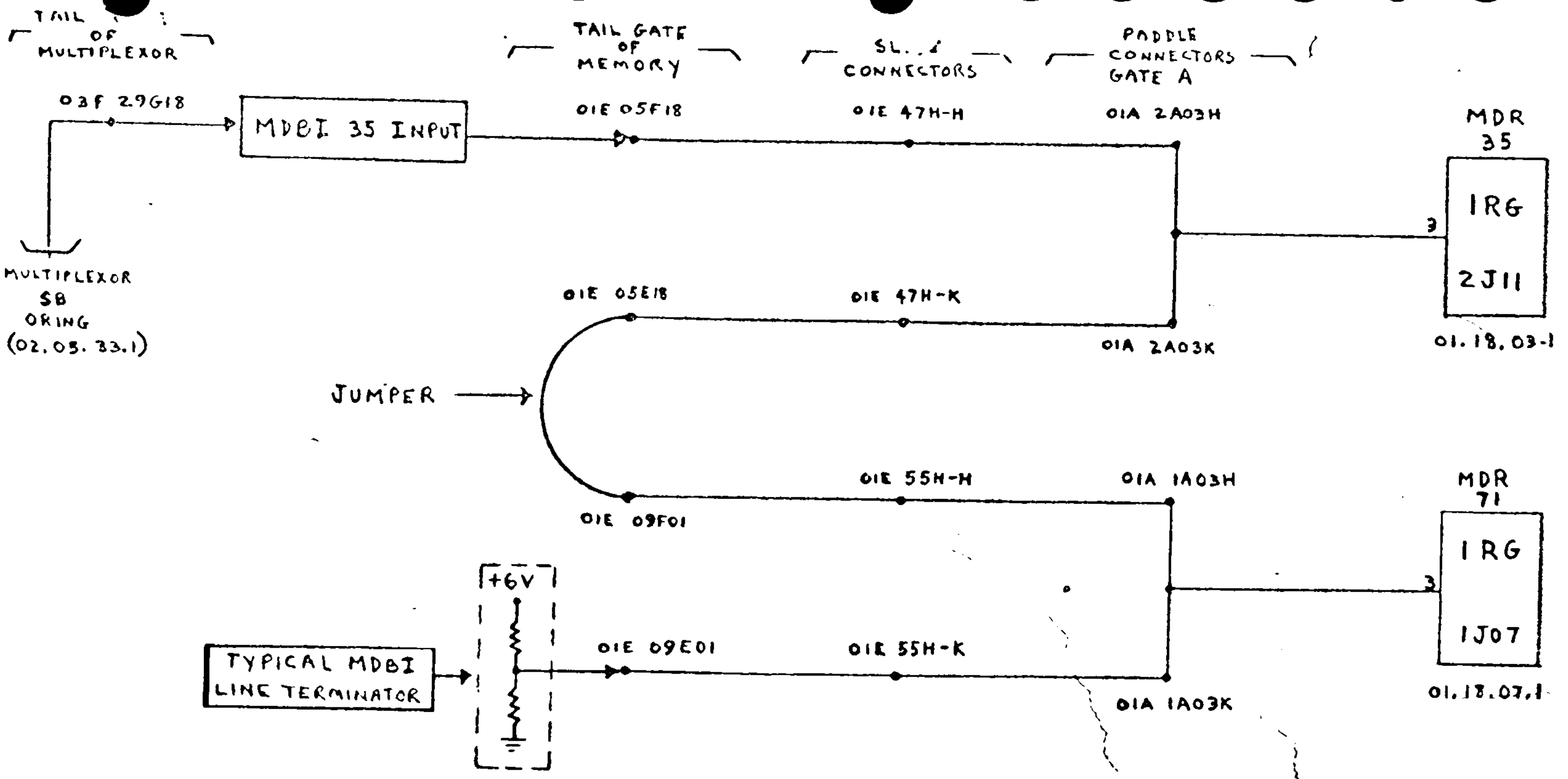


FIGURE 94A. TYPICAL MDBI CABLING

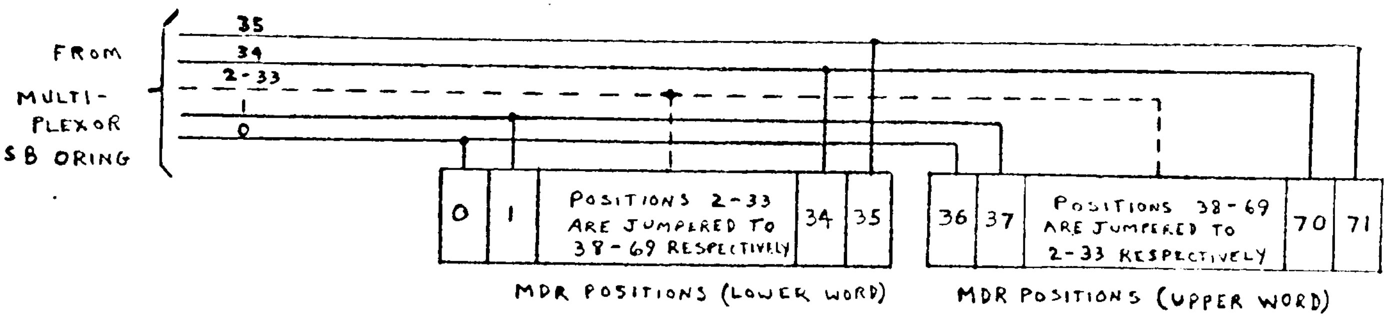
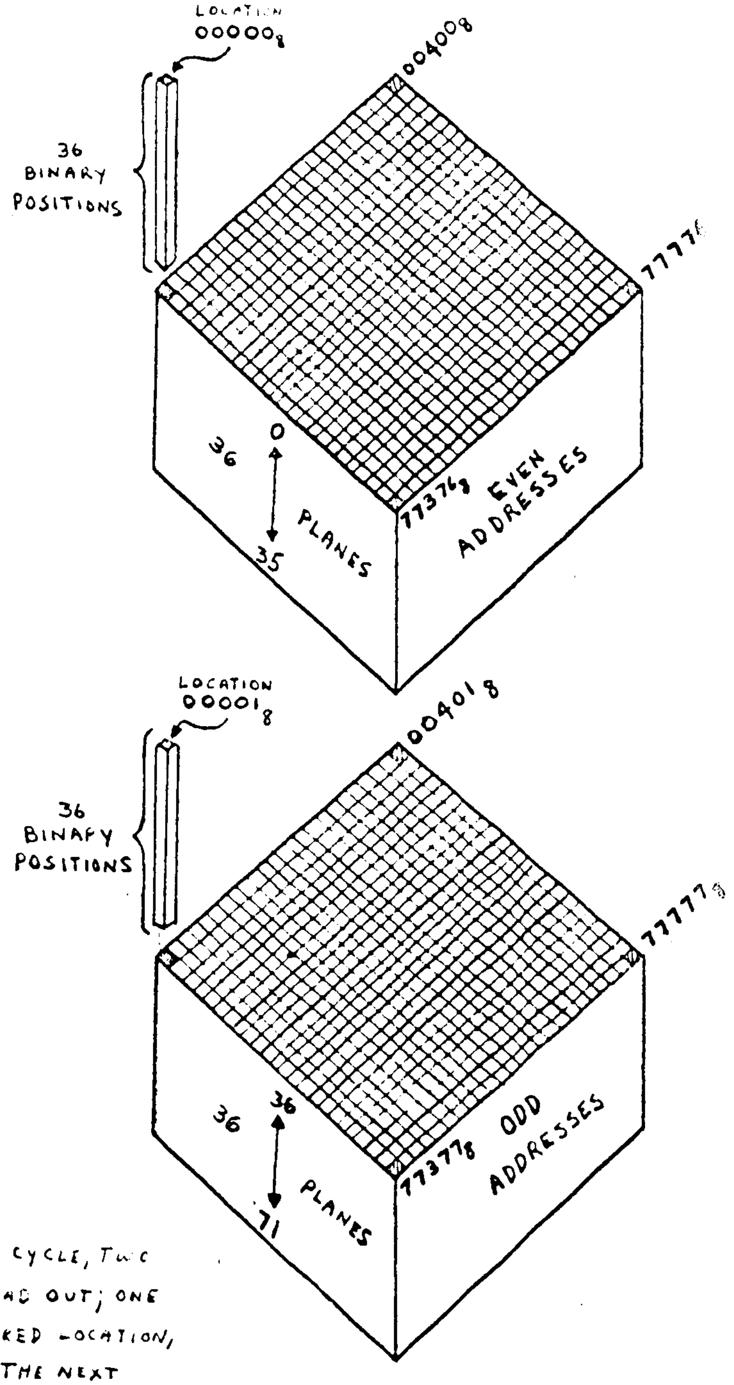

FROM MULTI-PLEXOR SB ORING
35
34
2-33
1
0
0
1
POSITIONS 2-33 ARE JUMPERED TO 38-69 RESPECTIVELY
34
35
36
37
POSITIONS 38-69 ARE JUMPERED TO 2-33 RESPECTIVELY
70
71
MDR POSITIONS (LOWER WORD)
MDR POSITIONS (UPPER WORD)

FIGURE 94B. MDBI TO MDR JUMPERING ARRANGEMENT

FIGURE 94. MDBI TO MDR CABLING (2.0-μsec MACHINE)

in location 40176_8 was also read out; if 40176_8 was requested, 00176_8 was brought out as well. Because the 7090 used only one word at a time, it did not matter that the word held in the nonselected location had no relation to the requested word. Both words were rewritten into storage on the second half of the cycle; a duplicate of the requested word having been sent to the computer, the nonselected word returning unused. It made no difference, then, that storage locations were numberd consecutively;locations 00000_8 through 37777_8 inclusive in lower memory (planes 0-35), and locations 40000_8 through 77777_8 inclusive in upper memory (planes 36-71).

In the modified 2.0-usec machine this storage arrangement has not been changed, i.e., to request a particular storage location, the same specific combination of 1's and 0's must be presented to the address circuitry of the 2.0-usec machine as was required by the 2.18-usec machine to read out that location. In addition, the two words read out will be displaced by 40000_8. However, by means of a crossover network in the multiplexor, the 7094 causes the 7302 to act as though lower memory contained only the even-numbered locations (00000_8 through 77776_8 inclusive) and upper memory only the odd-numbered locations (00001_8 through 77777_8 inclusive). For example, it has previously been shown that if location 00176_8 is requested, the nonselected word from location 40176_8, which is in the same relative position in upper memory, is also read out. However, as far as the 7094 computer is concerned, the nonselected word came from upper memory, therefore it must have come from an odd location. In addition, because the nonselected word was brought out with the word from location 00176_8, it must have come from the next highest odd location, or 00177_8. Figure 95 depicts core storage as seen by the 7094 computer. Notice that each even location in lower memory is shown in the same relative position as the



NOTE. ON ANY STORAGE CYCLE, TWO 36-BIT WORDS ARE READ OUT; ONE FROM AN EVEN-NUMBERED LOCATION, AND THE OTHER FROM THE NEXT ODD-NUMBERED LOCATION, E.G., LOCATIONS 0 AND 1.

FIGURE 95. STORAGE ARRANGEMENT (AS SEEN BY THE PROGRAMMER)

next highest odd location in upper memory, e.g., locations 00000_8 and 00001_8. Bear in mind that the computer considers the first location in upper memory to be 00001_8, when actually it is 40000_8. Therefore, when the computer sends out an address requesting location 00001_8, this address must be converted to 40000_8 before it is presented to the 7302. This is the function of the crossover network in the multiplexor which will be discussed in detail later.

Unlike the 7090, the 7094 computer may request two instruction words on one storage cycle, one from an even-numbered location, and the other from the next highest odd-numbered location. The even instruction is gated out of the MDR first, and the odd instruction follows shortly after. While the even instruction is being processed by the computer, the odd instruction is examined (partially decoded) in the instruction-backup register (IBR) to determine whether or not it meets certain criteria for an overlap operation. If overlap is possible, the odd instruction may be acted upon during E or L time of the ~~the~~ even-numbered instruction, ~~cycle~~, because decoding has already been initiated. This substantially reduces (up to 50%) the number of instruction cycles required. If it is determined that an overlap operation is not possible, the register is reset, and the odd instruction destroyed. It is brought back on the following storage cycle and processed as a single instruction.

STORAGE ADDRESSING

The crossover network in the multiplexor enables all even addresses sent out by the computer to request locations in lower memory, and all odd addresses to request locations in upper memory. This is accomplished by gating computer address position 3 into position 17 of the 7302 MAR, and computer address position ~~3~~ 17 into MAR ~~17~~ 3. In this way all addresses sent out by the computer are made compatible with the storage arrangement of the 7302, and changes to the addressing circuitry of the 7302 have been avoided.

Figure96 is a logic diagram of the crossover network. During normal operation, the memory diagnostic switch on the 7151 Console Control Unit is open. As a result, the converter partially conditions $-A_2$ and $-A_3$, and disables $-A_1$ and $-A_4$. If computer address position 3 holds a 1, the output of $+O_1$ is minus, and conditioning of $-A_3$ is completed. This causes -MAR line 17 to come up, and with the coincidence of an active + memory select line, a 1 is set in MAR position 17 of the 7302. In the same manner, a 1 in computer address position 17 results in a 1 in MAR position 3 of the 7302.

In order to run memory diagnostics , the 7094 is placed in a 7090 mode, i.e., the crossover network must be disabled. This is the function of the memory diagnostic switch (Figure 96). When this switch is closed, the converter partially conditions $-A_1$ and $-A_4$, and disables $-A_2$ and $-A_3$. As a result, the crossover network is inoperative, and computer address positions 3 and 17 can now only affect MAR positions 3 and 17 respectively. The memory diagnostic switch operating in conjunction with the crossover network affects the system as follows:

Switch Open	Switch Closed
1. Comp. add. position 3 gates MAR 17.	1. Comp. add. position 3 gates MAR 3.
2. Comp. add position 17 gates MAR 3.	2. Comp. add. position 17 gates MAR 17.
3. Planes 0-35 contain all even-numbered locations.	3. Planes 0-35 contain all locations below 40000_8.
4. Planes 36-71 contain all odd-numbered locations.	4. Planes 36-71 contain all locations above 37777_8.

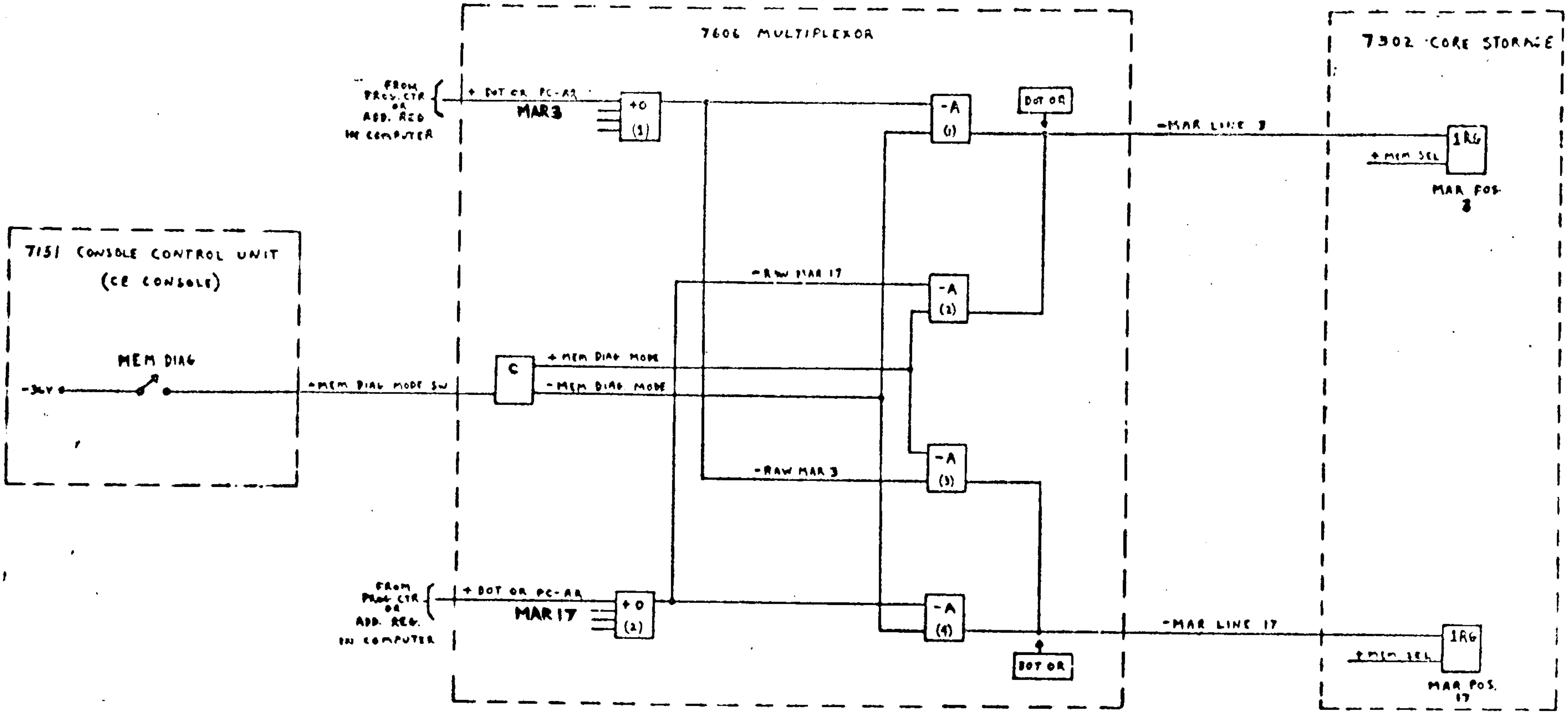


FIGURE 8. MAR 3 - MAR 17 CROSSOVER CONTROL

202,7

The block diagram in Figure 97 illustrates addressing procedures during an instruction-overlap operation. In the example, the computer sends out address 00176$_8$. This address passes through the multiplexor, and enters the MAR unaltered by the crossover networkbecause positions 3 and 17 both hold 0's. The MAR sends positions 4 through 17 to the core selection circuits, and MAR 3 to the gate-out circuits. As a result of the specific combination of 1's and 0's in positions 4-17, the instructions held in locations 00176$_8$ and 40176$_8$ are read out of core and placed in the MDR. Because of the 0 in MAR 3, the lower word (00176$_8$) is gated out of the MDR and sent to the computer. Shortly thereafter, the "split DOG" line comes up and gates the upper word (40176$_8$) to the computer (the modification to the gate-out circuits that permits two words to be sent out on one storage cycle will be covered later in "Gate-Out Circuits"). Remember that the computer considers storage location 40176$_8$ to be the next highest odd location, 00177$_8$.

While the instruction from location 00176$_8$ (even) is being processed by the computer, the instruction from 40176$_8$ (odd) is partially decoded in the IBR to determine whether overlap operations are possible during the current storage cycle. Assuming that overlap is not possible, the odd instruction is destroyed in the IBR. A second storage cycle is necessary, therefore, because the odd instruction must be brought back to the computer and processed as a single instruction.

Because it requires that only the odd instruction be returned, the computer sends out address 00177$_8$. Note that 00177$_8$ actually corresponds to storage location 40176$_8$, and must be presented to the 7302 as such if the same odd instruction is to be returned to the computer. When address 00177$_8$ passes through the multiplexor, the 1 held in position 17 is gated into position 3 by the crossover network. Thus, the address that enters the MAR in the 7302 is 40176$_8$. Although locations 00176$_8$

NOTE: IT IS ASSUMED THAT COMPUTER INSTRUCTION-OVERLAP OPERATIONS WERE NOT POSSIBLE ON THE FIRST CYCLE, THEREFORE, A SECOND STORAGE CYCLE WAS NECESSARY TO RETURN THE ODD INSTRUCTION FOR PROCESSING.

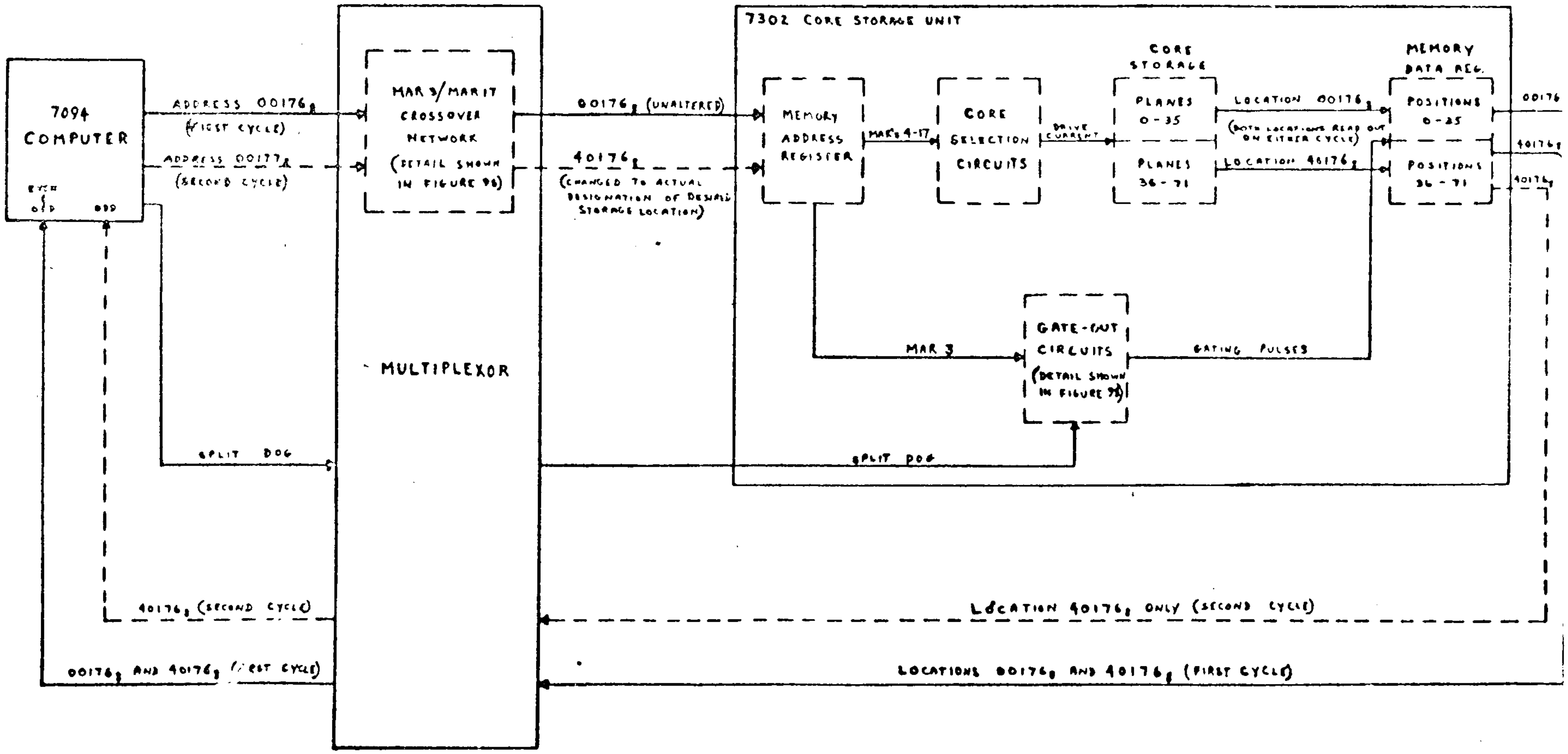

7094 COMPUTER
EVEN OR ODD
ODD
ADDRESS 00176_8 (FIRST CYCLE)
ADDRESS 00177_8 (SECOND CYCLE)
SPLIT DOG
MAR 3/MAR 17 CROSSOVER NETWORK (DETAIL SHOWN IN FIGURE 98)
MULTIPLEXOR
00176_8 (UNALTERED)
40176_8 (CHANGED TO ACTUAL DESIGNATION OF DESIRED STORAGE LOCATION)
7302 CORE STORAGE UNIT
MEMORY ADDRESS REGISTER
MAR 4-17
CORE SELECTION CIRCUITS
DRIVE CURRENT
CORE STORAGE
PLANES 0-35
PLANES 36-71
LOCATION 00176_8
(BOTH LOCATIONS READ OUT ON EITHER CYCLE)
LOCATION 40176_8
MEMORY DATA REG.
POSITIONS 0-35
POSITIONS 36-71
00176
40176_8
40176_8
MAR 3
GATE-OUT CIRCUITS (DETAIL SHOWN IN FIGURE 98)
GATING PULSES
SPLIT DOG
LOCATION 40176_8 ONLY (SECOND CYCLE)
LOCATIONS 00176_8 AND 40176_8 (FIRST CYCLE)
40176_8 (SECOND CYCLE)
00176_8 AND 40176_8 (FIRST CYCLE)

FIGURE 97. ADDRESSING PROCEDURES FOR INSTRUCTION-OVERLAP OPERATIONS

and 40176_8 are both subsequently read out of core, the 1 held in MAR 3 causes the gate-out circuits to select and gate to the computer only the word from upper memory (40176_8). The split DOG line has no effect on the lower word.

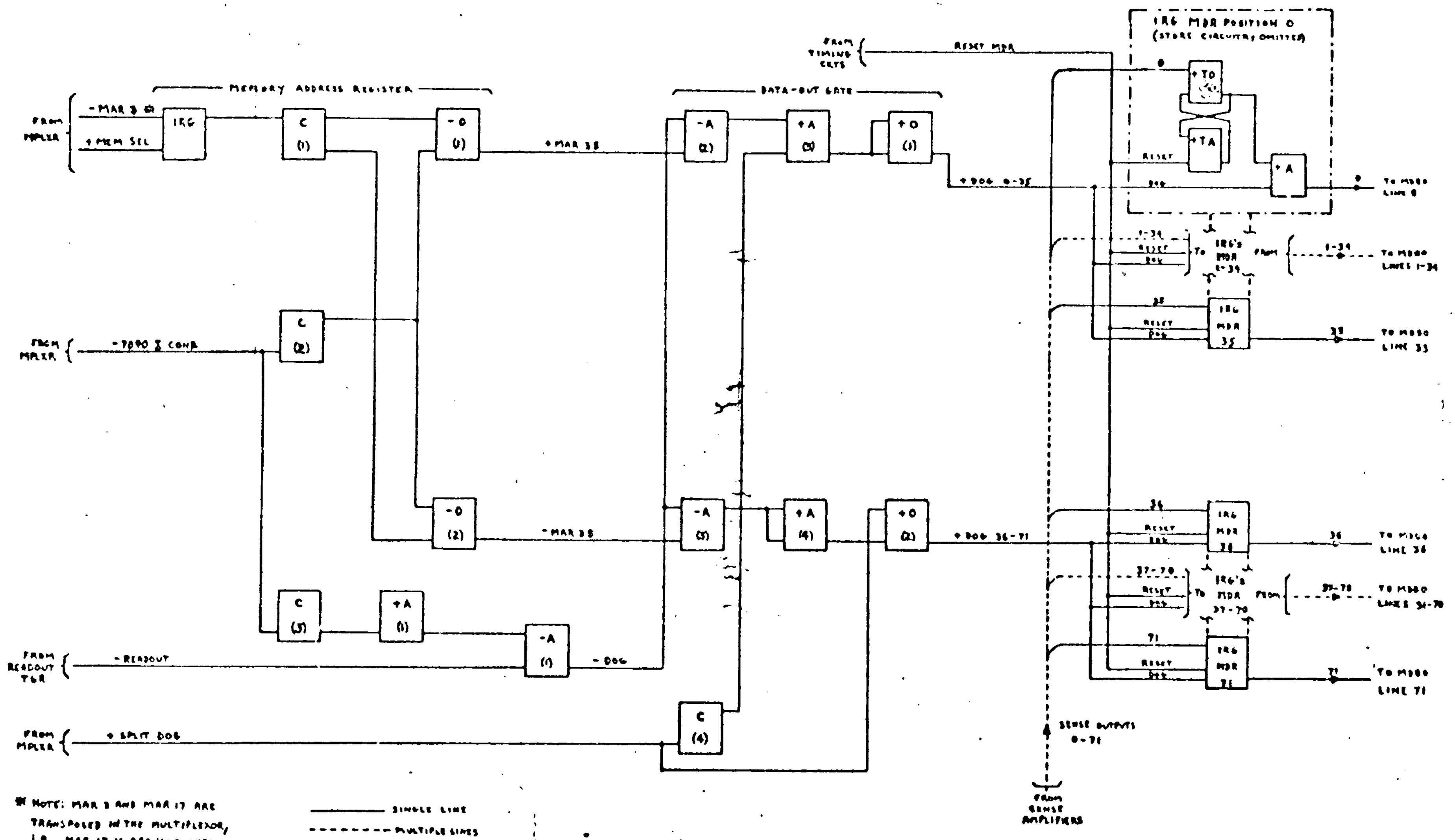
GATE-OUT CIRCUITS

It was stated earlier that the 2.0-usec machine is capable of gating out two 36-bit words on one storage cycle; one from an even location, and the other from the next-highest odd location. This is the result of a modification to the data-out gate circuits, and to the presence of the "split DOG" signal from the computer.

Assume that an address has been presented to the 7302, and by virtue of the combination of 1's and 0's in positions 4-17, the two words sharing the designated storage address have been placed in the MDR (Figure 98). Assume also that MAR position 3 holds a 0. Because of the 0 in MAR 3, the -MAR 3 input to the 1RG circuit is plus, therefore, its output is also plus. As a result, the the minus out-of-phase output of C_1 causes the in-phase output of $-O_1$ to be minus. Both inputs to $-A_2$ are minus. Conversely, both inputs to $+A_3$ are plus (split DOG is not active at this time). The resulting plus output of $+O_1$ is applied to the output AND circuits of MDR positions 0-35, and the word from lower memory (even) is gated out to the computer.

Shortly after the even word is gated out, the split DOG line comes up and causes the output of $+O_2$ to go positive. This output completes the conditioning of the output AND's associated with MDR positions 36-71, and the upper memory word (odd) is gated out to the computer.

When MAR 3 holds a 1, the output of the 1RG is minus, and the out-of-phase output of C_1 is plus. Because both inputs to $-O_1$ are plus, its output remains plus throughout the entire cycle, thus preventing the even word from being gated to the computer. Notice, however, that both inputs to $-A_3$ are minus, insuring a plus



* NOTE: MAR 3 AND MAR 17 ARE TRANSPOSED IN THE MULTIPLEXOR, I.E., MAR 17 IS BROUGHT INTO THE 7302 AS MAR 3.

FIGURE 98. GATE-OUT CIRCUITS (2.0 μSEC MACHINE)

output from $+O_2$ and gating of the odd word to the computer. Fromm the foregoing discussion it can be concluded that:

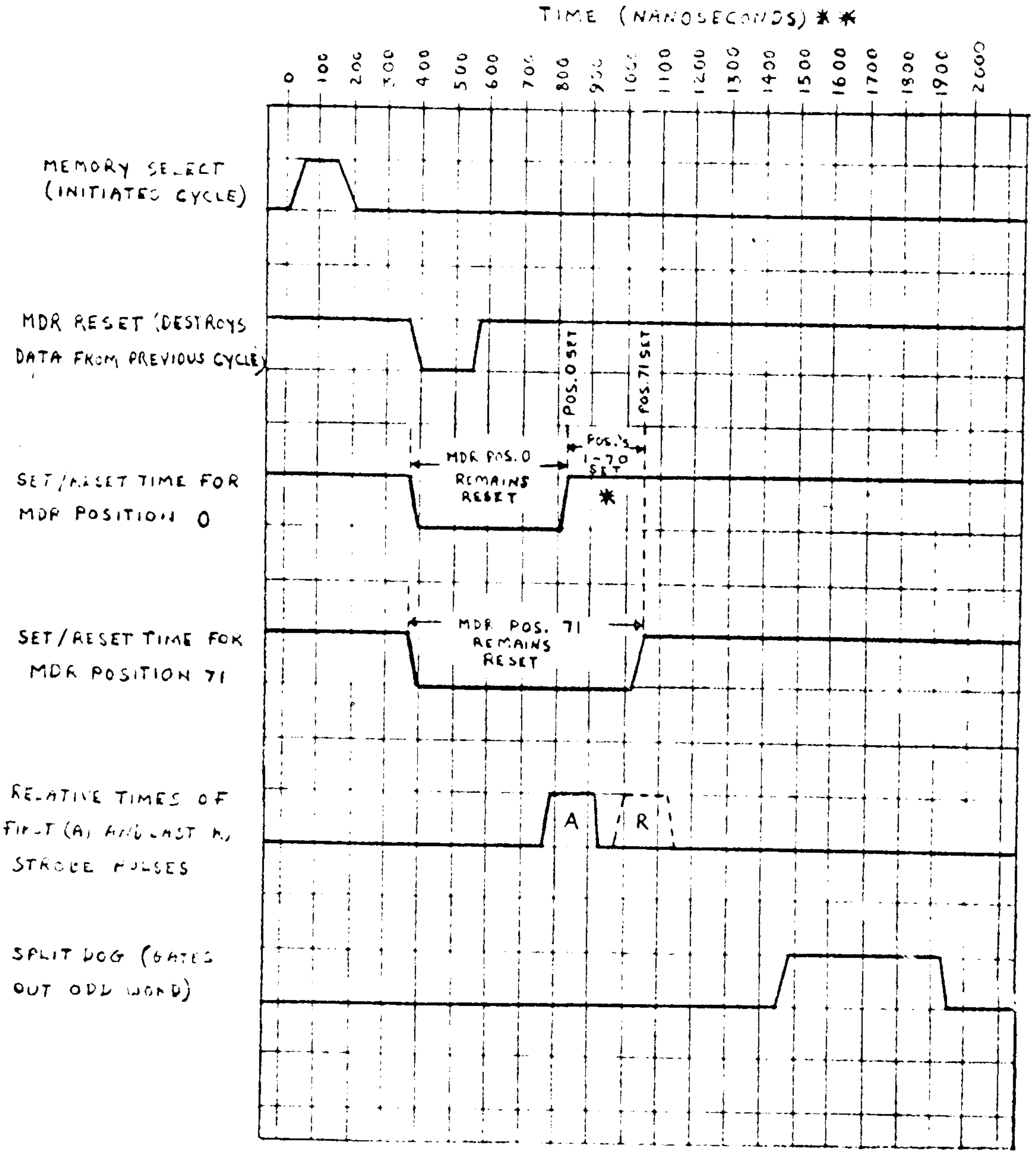
1. When MAR 3 holds a 0, two words may be gated to the computer during the storage cycle.

2. When MAR 3 holds a 1, only the odd word can be gated out.

The timing chart (Figure 99) gives the gate-out times for MDR positions 0-71, with respect to the first and last strobe pulses. All timings are approximate, and are referenced to the rise of memory select. All MDR positions are reset at 400 ns, therefore, no data exists in the MDR until the first plane is strobed at 800 ns. It is assumed that data from the entire 71 planes will have been read into the MDR by 1050 ns, which corresponds to the last strobe pulse. If MAR 3 holds a 0, the data from MDR positions 0-35 (even word) is placed on the MDBO lines and sent to the computer by 940 ns. At 1500 ns, the split DOG line comes up and places the odd word on the MDBO lines. If MAR 3 holds a 1, only the odd word is placed on the MDBO lines, and this is considered accomplished at 1050 ns.

7094-7302 ADDRESS CORRELATION

Half of the possible 32,768 different address combinations from the computer will have been altered by the crossover network in the multiplexor before being presented to the MAR in the 7302. This will be true whenever positions 3 or 17 (but not both) hold a 1. Therefore, the address exhibited by the MAR indicators on the CE test panel may not be representative of the address originally sent from the computer. To correlate any address shown on the CE test panel with the computer address required to produce it, use the following chart:


TIME (NANOSECONDS) **
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
1700
1800
1900
2000
MEMORY SELECT (INITIATES CYCLE)
MDR RESET (DESTROYS DATA FROM PREVIOUS CYCLE)
POS. 0 SET
POS. 71 SET
MDR POS. 0 REMAINS RESET
POS.'s 1-70 SET
*
SET/RESET TIME FOR MDR POSITION 0
MDR POS. 71 REMAINS RESET
SET/RESET TIME FOR MDR POSITION 71
RELATIVE TIMES OF FIRST (A) AND LAST (R) STROBE PULSES
A
R
SPLIT DOG (GATES OUT ODD WORD)

* NOTE: IF MAR 3 IS 0, THE LOWER WORD (0-35) IS PLACED ON MDR LINES BY 940 ns. SPLIT DOG THEN GATES THE UPPER WORD (36-71) AT 1500 ns. IF MAR 3 IS 1, THE UPPPER WORD IS PLACED ON MDR LINES BY 1050 ns. THE LOWER WORD CANNOT BE GATED OUT.

** NOTE: ALL TIMES ARE APPROXIMATE.

FIGURE 99 GATE-OUT TIMINGS (2.0 μs MACHINE)

MAR Indicators	Address From Computer
00000_8 to 37776_8 (All Even).	As indicated.
00001_8 to 37777_8 (All Odd).	Subtract 1, and add 40000_8.
40000_8 to 77776_8 (All Even).	Subtract 40000_8, and add 1.
40001_8 to 77777_8 (All Odd).	As indicated.

To determine the address that will be presented to the 7302 MAR for any address sent out of the computer, use the following table:

Address From Computer	Address Presented to MAR
00000_8 to 37776_8 (All Even).	Same.
00001_8 to 37777_8 (All Odd).	Subtract 1, and add 40000_8.
40000_8 to 77776_8 (All Even).	Subtract 40000_8, and add 1.
40001_8 to 77777_8 (All Odd).	Same.

SENSE AMPLIFIERS

The isolating pulse transformers used in the sense amplifiers of the 2.18-usec machine are too slow for 2.0-usec operation. In the modified machine, these have been replaced with a suitable type. Figure 49 shows a portion of a typical sense amplifier. This isolating pulse transformer is T1, and the differential amplifier that feeds the transformer, consists of transistors T1 and T2.

A diode clipping network has also been added. This network limits the noise induced in the amplifiers by inhibit current. Recall that the sense and inhibit windings in a core plane are wired perpendicular to each other to reduce the noise induced in the sense windings by the Z drivers. However, the noise that gets through to the sense amplifiers is troublesome. Even though the sense amplifiers are not used during Z time (rewrite), the voltage induced in the sense windings makes it difficult for the amplifiers to recover before read time on the following cycle.

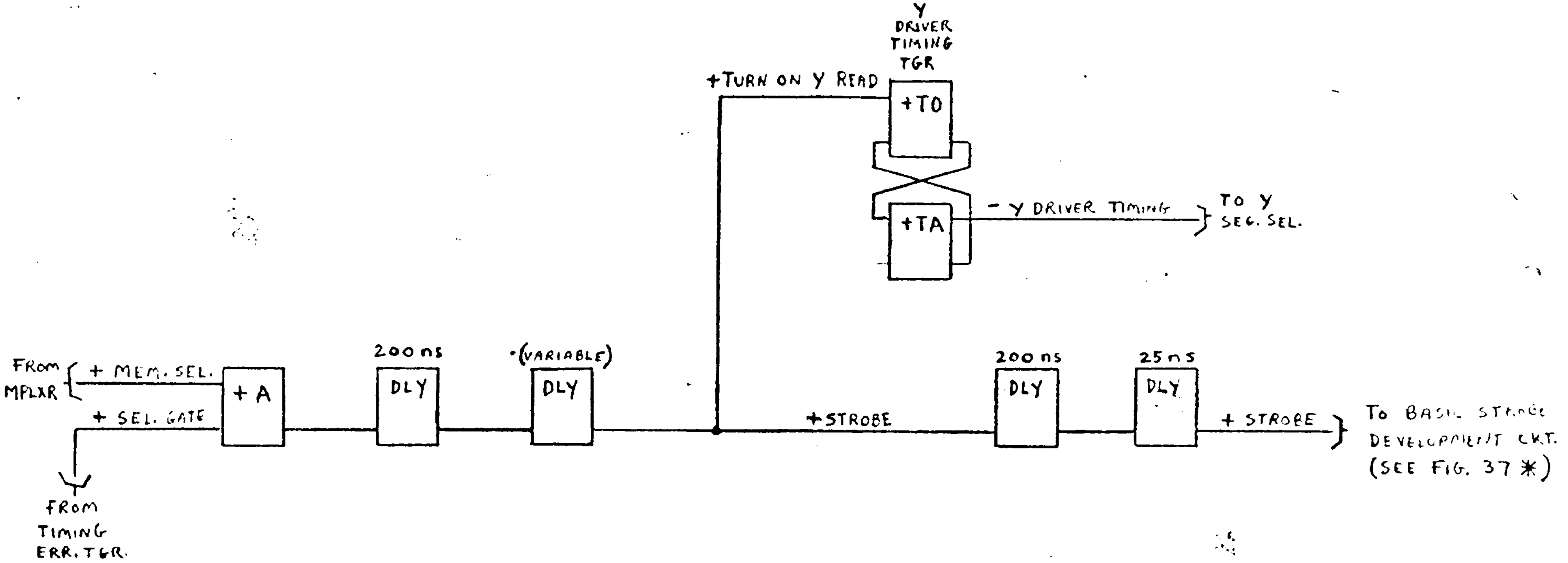
These modifications to the sense amplifiers have resulted in a change in the card type designation. The ZN-- cards (P/N 395454) in the 2.18-usec machine have been changed to DDQ- (P/N 380081) in the 2.0-usec machine (Systems 01.17.00 through 01.17.13.1).

STROBE CIRCUITS

The manner in which the strobe circuits are turned on in the modified 7302 does not differ~~s~~ from that of the 2.18-usec machine. ~~Recall that~~ strobing in the 2.18-usec machine is initiated by the strobe level pulse. This pulse, which results from a matrix switch flipping in the selected Y switch segment, is applied to the basic strobe development circuit shown in Figure 37. In the modified machine, however, strobing is initiated by the same circuit that turns on the Y driver-timing trigger. This is shown in Figure 100.

The combination of + memory select from the multiplexor and + select gate from the timing error trigger produce a plus-level pulse at the in-phase output of the +AND circuit. After a delay of 200 nanoseconds, the + strobe pulse is developed. This pulse is applied to the basic strobe development circuit (Figure 37) at the input to the converter (line A), replacing the strobe level pulse and DLY 1. Operation of the remainder of this circuit,as well as the strobe generation circuits (Figure 38), is the same as that described for the 2.18-usec machine.

This change to the strobe circuits is significant in that the basic strobe is no longer dependent on the true-one driver for the selected Y switch segment. This greatly expedites the process of determining the cause of missing or late strobes. For example, if the strobe pulses to one particular group of sense amplifiers is missing, it is safe to assume that the trouble is probably in that portion of the strobe



* NOTE: IN THE 2 μ SEC. MACHINE, THE + STROBE LINE CONSTITUTES THE INPUT TO THE CONVERT BLOCK (LINE (A) IN FIGURE 37), REPLACING THE STROBE LEVEL PULSE AND DLY 1. IN THE 2.18 μ SEC MACHINE, STROBE TURN-ON IS A RESULT OF AN OUTPUT FROM THE SELECTED Y MATRIX SWITCH SEGMENT.

FIGURE 100. STROBE TURN-ON CIRCUIT (2 μ SEC MACHINE)

generation circuits responsible for providing strobes to the sense amplifier group concerned. If the strobes to all sense amplifiers is missing, the basic strobe development circuit is the probable cause. In addition, early or late strobes may now be traced directly to the strobe generation circuits.

MAIN ADDERS

The adders in the 7094 CPU are made up of the basic building block shown in Figure 205-1. The adders perform all arithmetic functions and are involved in most internal transfer functions within the computer. Note in Figure 205-1 that inputs to the basic adder circuit may be from the accumulator or from the storage register. Other input lines may be selected by programming. Outputs from the adders usually go to the accumulator although, again, programming may select alternate routes.

The adders are comprised of 39 individual adder units, or bit positions. These include 1 through 35, Q, P, 9P and 9Q. Positions Q and P are used for overflow which might occur during certain arithmetic operations. Positions 9Q and 9P are used during floating point arithmetic.

Individual Adder

The adder unit shown in Figure 205-1 is for bit position 33. The four outputs, singly, or in combination, indicate the sum of the incoming bits, whether a carry has been developed to the next high order bit position and whether the group of adders (bit positions 31 through 35) will result in a carry to the next higher group. The combination of bit and group carry indications constitute the lookahead capability of the adders.

Lookahead Feature

To facilitate lookahead, the adders have been divided into ten groups, according to the following chart. A carry-out of each group and the levels required for lookahead are developed in two sequential logic blocks of OR and AND circuits (Figure 205-3). Groups one through five are combined into a section and the section carry lookahead is developed in two sequential blocks of AND and OR circuits. The outputs of groups one through five, and six through nine are combined in an AND circuit to develop the carry into group ten

(Figure 205-4). Provision has also been made for a generate output to be developed from groups six through nine, indicating that addition within the adders is going to result in a carry into group ten. The total number of logic blocks involved in carry development or lookahead is six. With 20 nanoseconds delay encountered in each logic block, the total delay between the time a one is injected into position 35 until it is felt in group ten is 120 nanoseconds. Without this lookahead capability the carry one would have to ripple through each bit position, resulting in a two-level delay of 35 times 40 or 1400 nanoseconds. The speed of the 7094 cannot tolerate this delay.

	Bit Positions	Group Number
Section	35, 34, 33, 32	1
	31, 30, 29, 28	2
	27, 26, 25, 24	3
	23, 22, 21, 20	4
	19, 18, 17, 16	5
Section	15, 14, 13, 12	6
	11, 10, 9, 9P	7
	9Q, 8, 7, 6	8
	5, 4, 3, 2	9
	I, P, Q	10

Chart of grouping of bit positions into groups and sections.

Lookahead and Carry

To understand the development of lookahead and carry it is necessary to understand the meaning of the terms propagate and generate. As used in relation to the individual bit position, refer to Figure 205-1, the term propagate means that the sum of the inputs (exclusive of the carry input) is at least one, i. e., at least one of the inputs must be a one (the other may be a zero). Accordingly the +P Propagate 33 line will be up whenever the conditions of the exclusive OR (a 1 and a 0) at the adder input are met. The chart on Figure 205-1 shows this to be the case. However, the chart also shows that if both inputs are one's, the propagate line will again be up. By definition the sum must be at least one, therefore, a sum of more than one (1+1 = 0 and 1 carry) will result in an up level on the propagate line.

With a one on both inputs, a generate condition exists. The generate signal means that the sum of the adder inputs (exclusive of carry) must be two (1+1 = 10) and a carry to the next high order bit will be developed. The generate line must be high whenever both inputs to the adder are one's. Because the complement of the signal is used, the chart shows that the +P Complement Generate 33 line will be up for all conditions other than the addition of two one's and down for the generate condition.

The Complement X or 33 output develops the sum and carry of each bit. This should be read as "complement of the exclusive OR condition at the input of bit position 33". Therefore, when an exclusive OR exists at the adder input, i. e., a one and a zero, this line will be down (complement). Note that the sum is developed through a negative exclusive OR.

Assume the input from the storage register is a one, that from the accumulator is a zero and there is a carry from position 34 (+P Carry in 33 is high). With these conditions we would expect to have a sum of zero and a one carry to the next high order bit, position 32. Inputs to the negative exclusive OR are -P (+P Complement X or 33 is down) and +P (Carry in 33 is up) so the negative exclusive OR conditions have been met and the output will follow the sign of the function -- a negative output indicating a sum of zero.

Adder Carry

Refer to Figure 205-2 to see how the carry out of a bit position is developed. A carry into position 33 means that bit position 34 adder either added two one's from the primary inputs or added a one to a one carry from bit position 35. With either condition the three OR circuits into the AND circuit are conditioned and the carry output from the AND logic block is high, indicating a carry into position 32.

The foregoing discussion illustrates the arithmetic function of addition and carry within a group of four adder bit positions. Figure 205-3 shows these same propagate and generate outputs develop carries outside the group, as part of the lookahead function. The propagate output will be down for the condition when both inputs to the adder are zero and up for all other conditions (1, 0 or 1, 1). The complement line will be up for 0, 0; down for 1, 0 or 0, 1; and up for 1, 1. With a 1 input to each of the four bit positions and a carry into position 35 logic circuitry causes the Generate G1 and carry into 31 outputs to be high.

Figure 205-3 also shows that a 1, 1 input to adder 32 must always result in a Generate G1 signal and carry into 31.

Group and Section Carry

Figure 205-4 illustrates how the group generage signals are tied together to advance the lookahead into group ten which contains the 1 bit and two overflow powitions (P and Q). For a carry into group ten there must be a generate condition in bits 2 through 15 OR a section carry from bits 16 through 35 AND a propagate signal from bits 2 through 15. These logic circuits then predict that if there is a section carry out and each bit position in adders 2 through 15 has a sum of at least one, there will be a carry into group ten. Also, the logic circuits predict that if the sum in each of the adder positions 2 through 15 is two, there will be a carry into group ten.

Special applications of lookahead and carry will be covered in those sections on multiply, divide and floating point operation.

When Registers Hold (No Carry):

SR	0	1	0	1
AC	0	0	1	1

Signals Are;

−	+	+	+	Propagate 33
+	+	+	−	Comp Gen 33
+	−	−	+	Comp Xor 33
−	+	+	−	AD 33 (Sum)

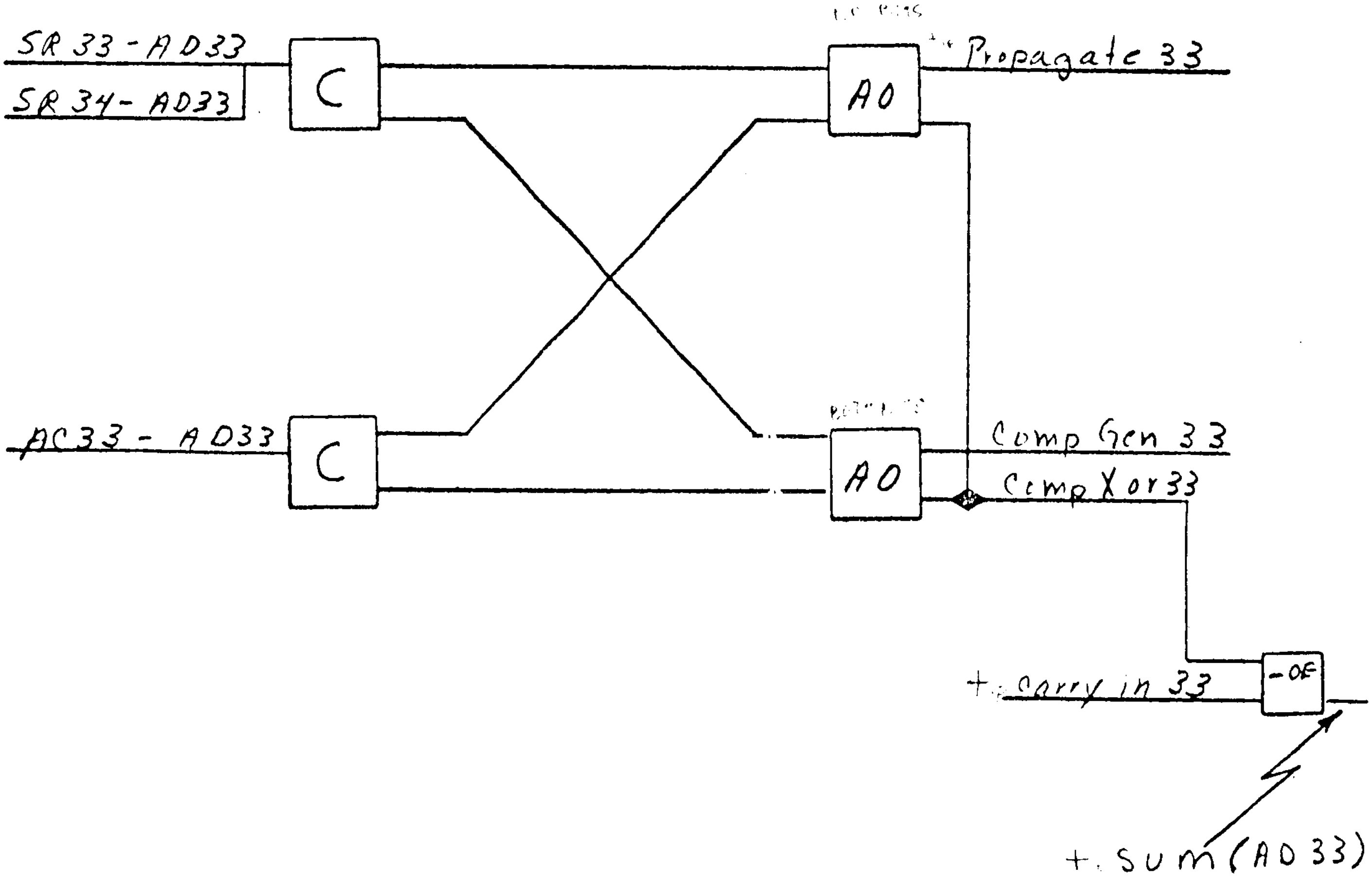


Figure 205-1. Adder Detail (Bit 33)

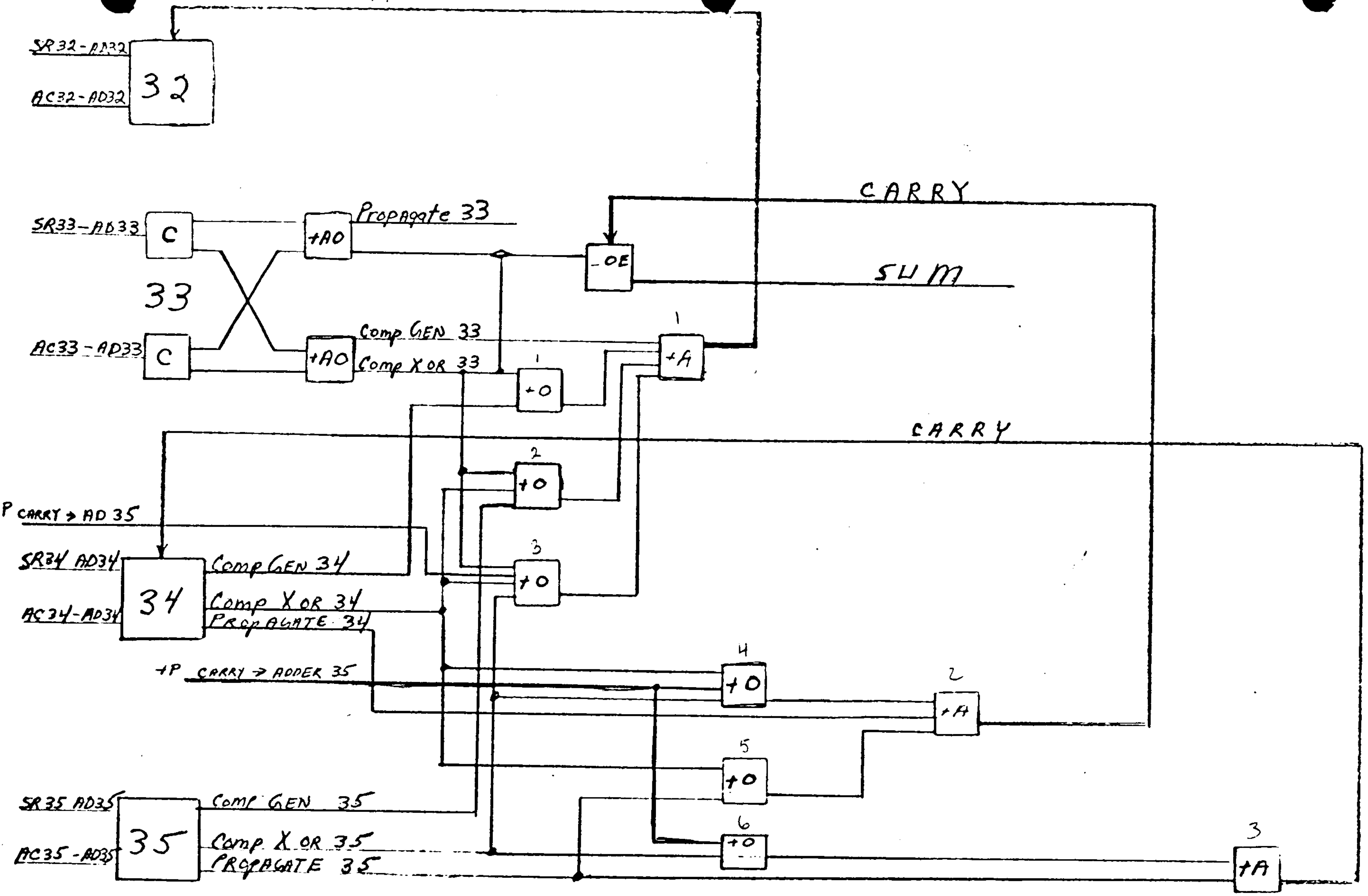


Figure [illegible]-2 Lookahead and carry development within a group

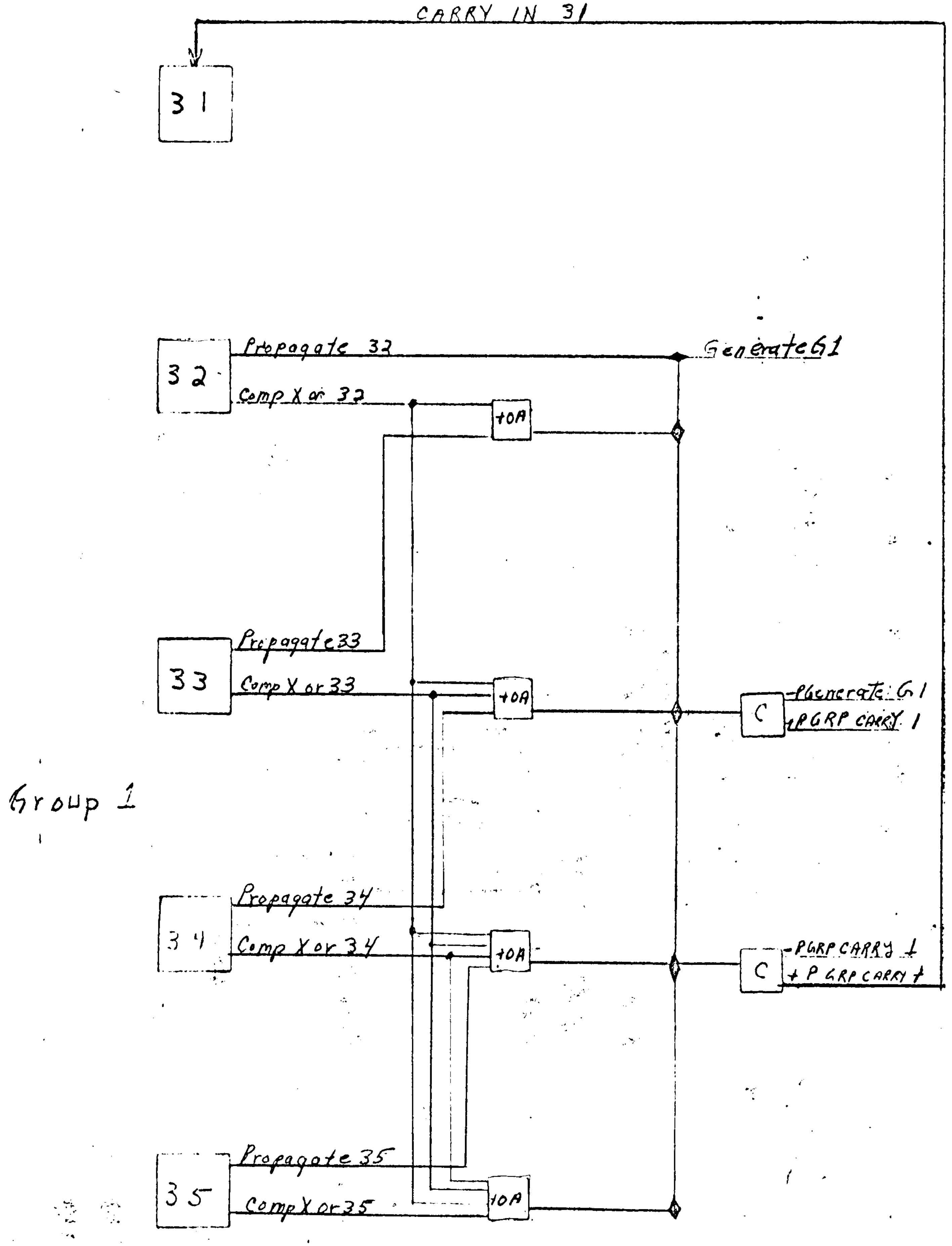


Figure 205-3 Group Lookahead and Carry

205.7

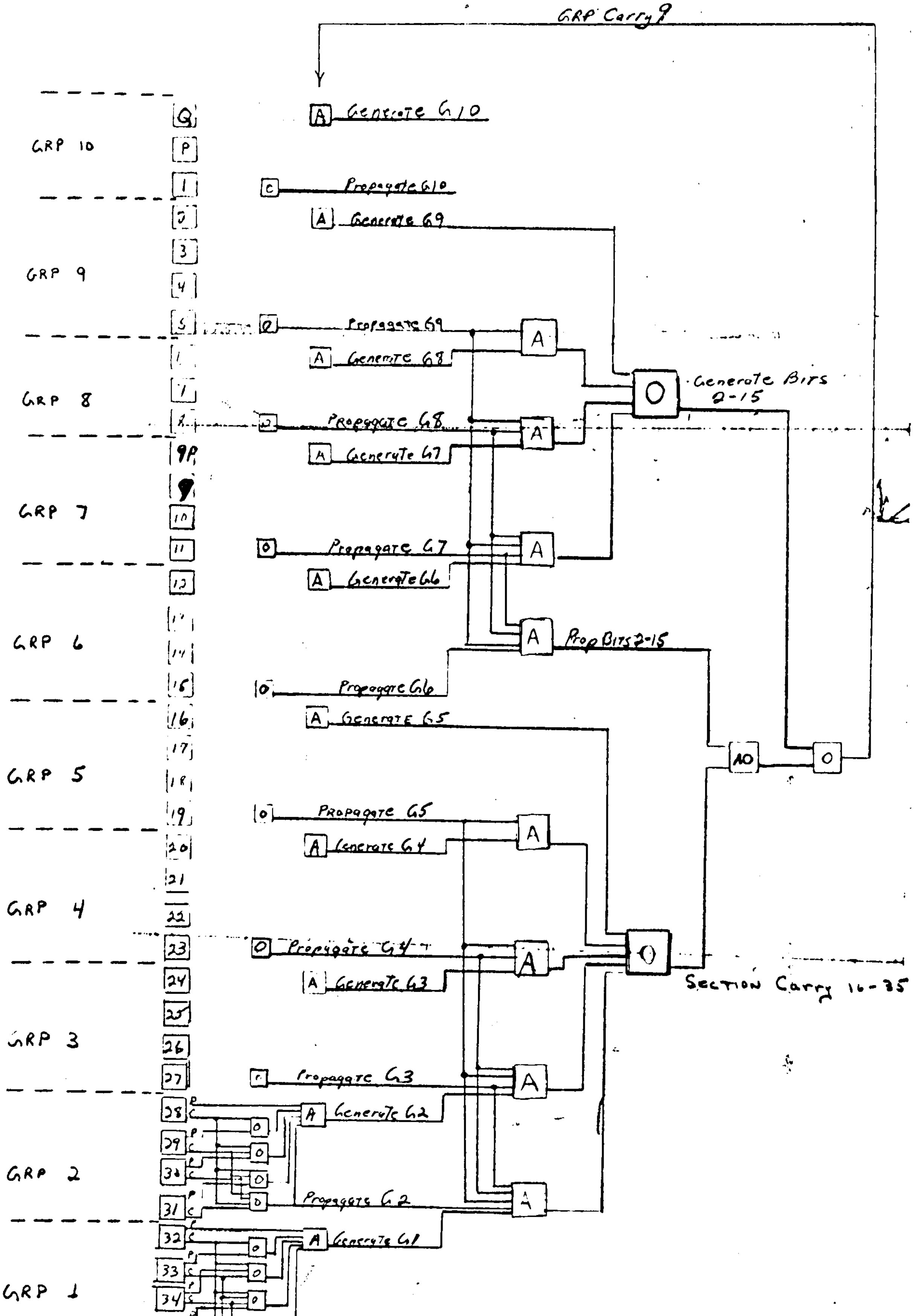
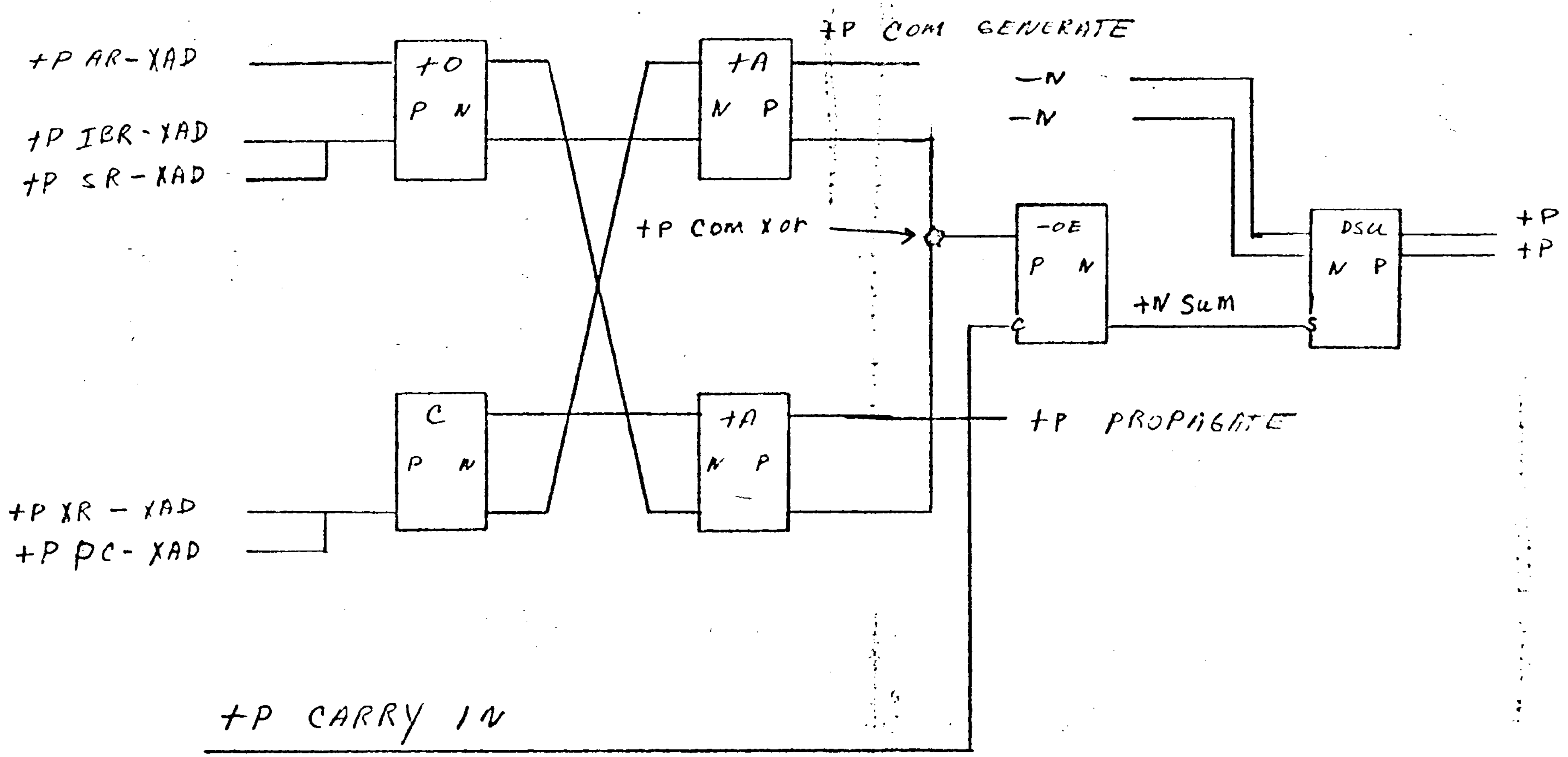

GRP Carry 9
Generate G10
Propagate G10
Generate G9
Propagate G9
Generate G8
Propagate G8
Generate G7
Propagate G7
Generate G6
Propagate G6
Generate G5
Propagate G5
Generate G4
Propagate G4
Generate G3
Propagate G3
Generate G2
Propagate G2
Generate G1
Generate Bits 2-15
Prop Bits 2-15
Section Carry 16-35
GRP 10
GRP 9
GRP 8
GRP 7
GRP 6
GRP 5
GRP 4
GRP 3
GRP 2
GRP 1
Q
P
1
2
3
4
5
6
7
8
9P
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
A
O
AO

FIGURE 205-4

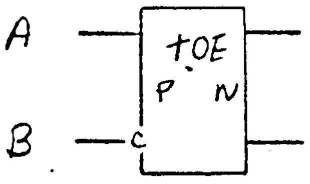


208.1

MFS 5/15/62

Cascode Exclusive Or

"Plus"



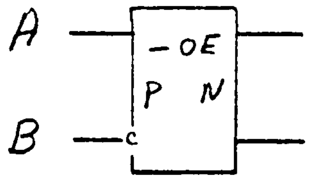
out φ $D = \bar{A}\bar{B} + AB$ or $\overline{A \forall B}$

IN φ $C = A\bar{B} + \bar{A}B$ or $A \forall B$

	T_1	T_2	T_3	T_4	
A	−	+	−	+	
B	−	−	+	+	
C	−	+	+	−	IN φ
D	+	−	−	+	out φ

IN φ output + when ether input + but not both together

"Minus"



out φ $D = (A+B)(\bar{A}+\bar{B})$ or $A \forall B$

IN φ $C = (\bar{A}+B)(A+\bar{B})$ or $\overline{A \forall B}$

	T_1	T_2	T_3	T_4	
A	−	+	−	+	
B	−	−	+	+	
C	+	−	−	+	INφ
D	−	+	+	−	Outφ

IN φ output − when ether input − but not both together

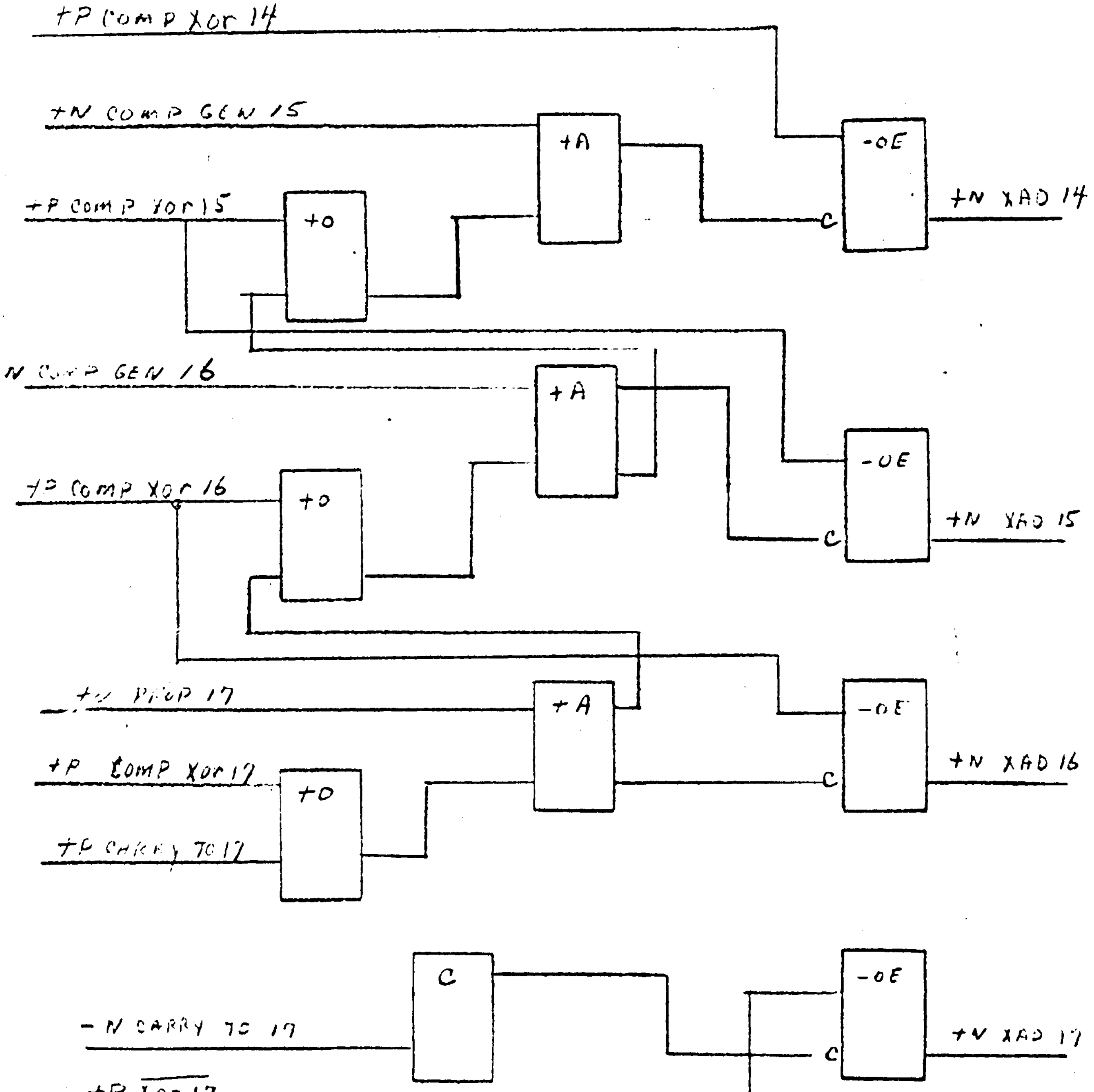
208.1A

~~203.5B~~

MFS 5/15/62

INDEX ADDER (M 17)

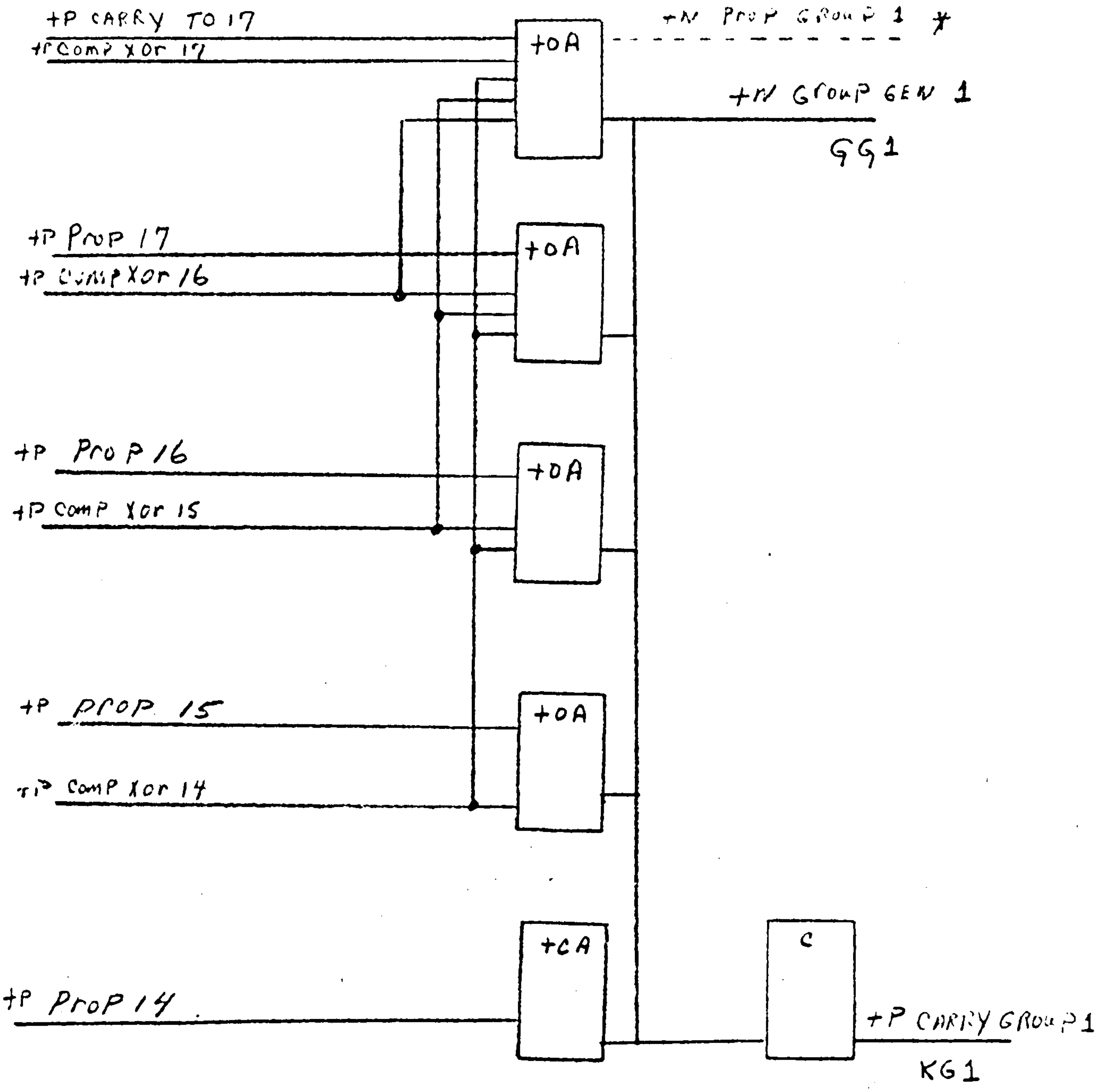
CARRY GENERATION WITHIN A GROUP



MFS 6/21

INDEX ADDER (14-17)

GROUP GENERATION

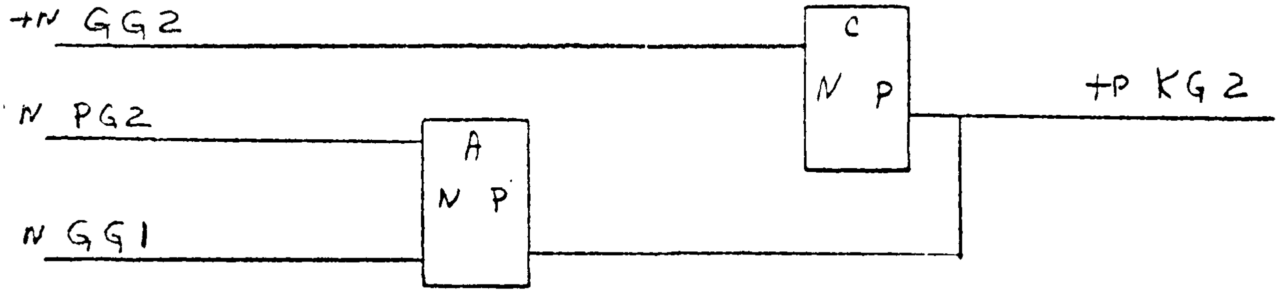


* ON OTHER GROUPS NOT ON GROUP 1

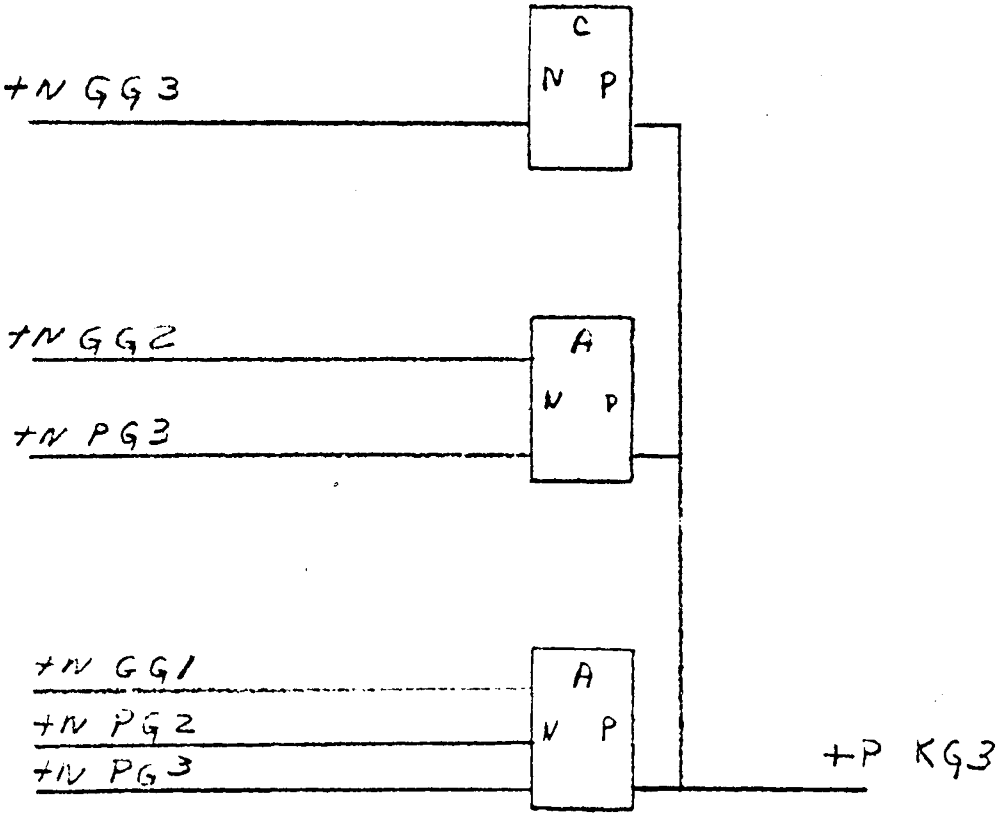
MFS 6/21/62

INDEX ADDER

CARRY GENERATION



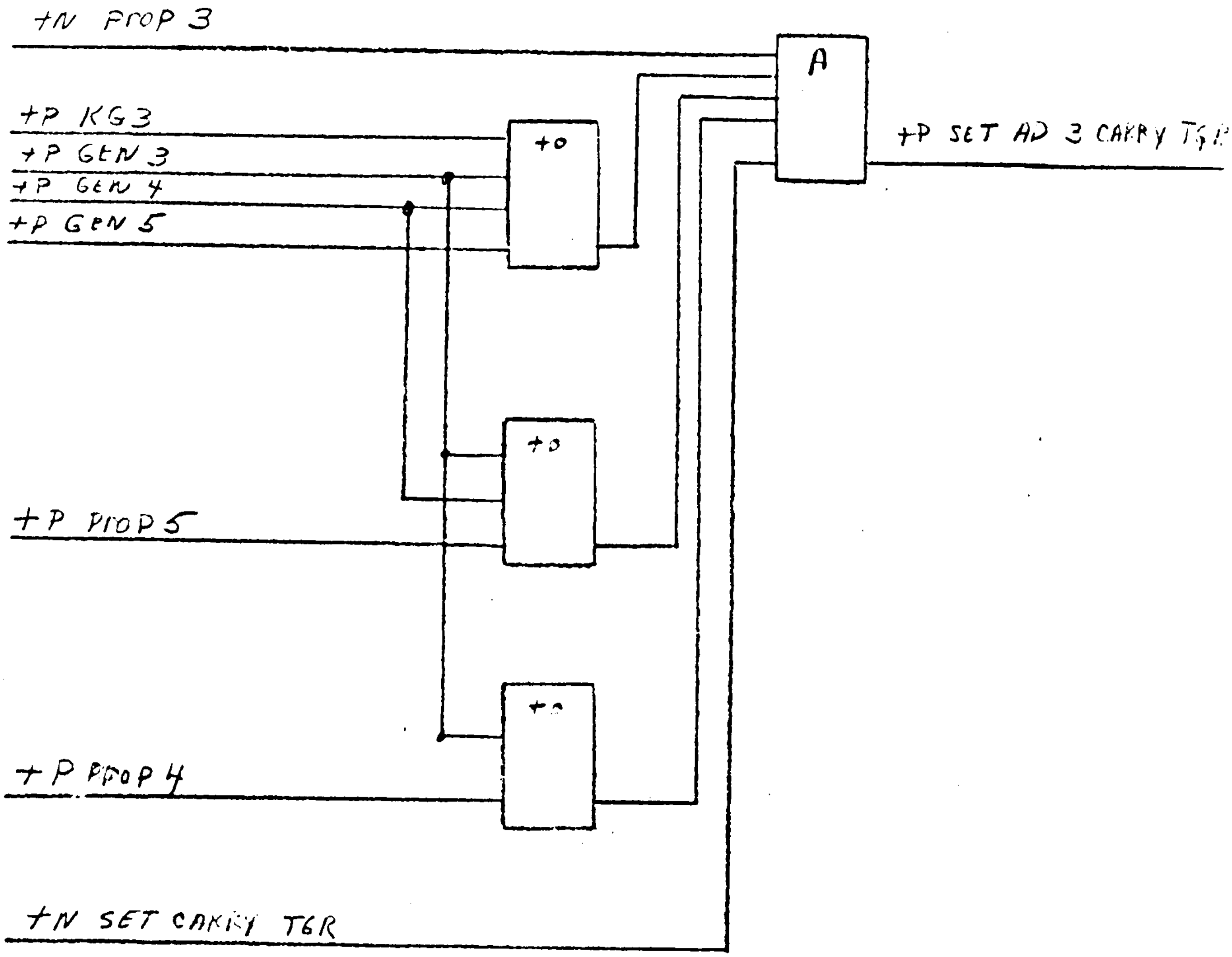
GROUP 2



MFS [illegible]

INDEX ADDER

AD 3 CARRY



±1

MFC 6/2/62

209 Index Registers (XR) Systems 03.05.00.1-03.05.14.1

There are seven XR's in the 7094. Each XR consists of 15 single triggers with a delayed output labeled 3-17. All seven XR's are identical in their operation and are used for address modification. Individual positions of all seven XR's have a common output. The outputs of the XR's are gated out by a specified tag (see section on tag registers) to the XAD's where the actual address modification is performed. The XR's are read out in their complement form and by addition of this complement number to an address, the address is effectively reduced by the contents of the XR.

There are many instructions which operate on the Index Registers, thereby making these registers useful programming tools for counting, word alteration, program loop control, and so forth.

Instruction Backup Register (IBR) Systems 03.08.00.1-03.08.05.1

The IBR is a 36 position trigger register whose positions are labeled S1-35. The IBR is used during Instruction Overlap operation and contains the next higher odd address which will be brought out if the instruction reference was to an even location. (See sections on DLA, SLA, and TLA for operation of the IBR during these operations). The IBR is loaded from the Multiplexor Storage Bus, with the odd address after the even address has been placed in the Storage Register. The IBR is also loaded from the Storage Register during Double Precision and also while performing the instruction ERA.

Outputs of the IBR are gated to the Program Register for decoding and also to the Tag Register for indexing. During TLA with TIX, TXI, TNS the IBR 3-17 is gated to the XAD. (See section on TLA.)

204 Accumulator Register (AC) (Systems 02.03.00.1-02.03.19.1)

The AC is a 39 position register each position consisting of a double latch (See section on 7094 double latches). The register positions are labeled (S) (Q) (P) (1-8) (9P) (9-35). Position S1-35 accommodates the word in standard operations. Positions (Q) and (P) are used as overflow positions because the sum of two 35 position numbers can be greater than 35 positions. Position (P) also holds the S bit of a word during logical operations. Position (9P) is also an overflow position which replaces the 9 overflow trigger which was used on previous systems.

The term accumulator is somewhat misleading since the register is not able to accumulate. The AC can contain only one word at a time. When adding, the AC and adders work as a unit to perform the addition, and the AC merely holds the result.

211 Multiplier - Quotient Register (MQ) Systems 02.04.00.1-02.04.09.1

The MQ is a 36 position register each position consisting of a double latch. (See section on 7094 Double Latch). The MQ receives its name from the functions it performs in the CPU. During a MPY operation it contains the Multiplier; after the MPY it contains the least significant half of the product. During a DIV, it contains the least significant half of the dividend, after the DIV it contains the Quotient. Due to its shift cell makeup the MQ has the ability of shifting left or right. In addition to this it also has the ability to perform a function called Ring Shift. This means bits can be shifted out of the S position and into MQ 35.

The entire MQ can be operated upon as is the case with the instruction STQ. In this case the entire MQ is gated to the Storage Reg and then to memory. There are instructions that only operate on portions of the MQ such as in CRQ where SR S1-5 are gated to the MQ 30-35. In this case only these positions are affected. During CRQ and CAQ MQ S1-5 are gated to the XAD 12-17. During Floating Point operations the ACC and MQ are shifted right and bits leaving ACC 35 are gated into MQ 9. There is another gating line which is only used during DFAD which gates MQ 35 to ACC 9 while shifting the MQ and ACC right. This is effectively a ring shift.

206

ADDRESS REGISTER (AR) SY[illegible] - 03.06.04.1

The Address Register is a 15 position trigger register whose positions are labeled (3 - 17). The contents of the Address Register are sent to the storage address register to set up the proper storage location to be used or in the case of indexing the contents may be gated to the Index Adders. The Address Register receives its information from the Storage Register, Index Adder or IBR. The Address Register is reset prior to reading new information into it. There is a delay block on the output of each position which is used during indexing to prevent you from affecting the XAD's while you are resetting. The Address Register and P. C. outputs are DOT ORE'D together and depending upon certain conditions being met will determine whether the PC or AR will be gated to Memory . Example - On successful transfers the AR will be gated to Memory. If the transfer was not successful then the P. C. contains the location of the next instruction and it will be gated to Memory.

2/2 PROGRAM COUNTER (PC) SYSTEMS 03. 06. 30. 1 - 03. 06. 34. 1

The Program Counter is a counter in name only due to the fact that when it is to be stepped the information in the counter is read into the XAD the counter is reset, a hot one is added to the XAD and the information is then placed back into the counter. The P. C. consists of 15 triggers labeled 3 - 17, with a delayed output to prevent the reset of the counter from affecting the contents of the XAD which are going to be placed back into the counter. The content of the program counter is normally stepped at each I7 time and gated back to the counter at 8 time and is used to tell the computer where in storage to obtain its next sequential instructions. Under these conditions the PC is sent directly to memory. When a successful transfer instruction is executed the new address is located in the Address Register which is sent to Mar. In order to change the contents of the P. C. the A. R. is gated to the XAD at 2 time and then gated to the P. C. at 3 time. At 7 time the contents of the P. C. are routed through the XAD's, stepped and brought back to the P. C. There are instructions which cause the computer to skip one or two instructions. These special conditions will be covered during the discussion of these instructions.

When performing an STL the compliment of the P. C. will be gated to the XAD's where it will be recomplimented and gated through the Storage Register to Memory.

213 PROGRAM REGISTER (PR) SYSTEMS 03.04.06.1 - 03.04.06.1

The program register is a ten position register, each position consisting of a single trigger. The program register receives the operation code of the instruction from the SB which can be brought up from a storage address, the Instruction Back-UP Register (See Section on IBR) or from the Entry Keys. The output of the Program Register is decoded to initiate and control the computer so it will execute the instruction. Positions (1-5) contain the primary operation part of the operation code and positions (6 - 9) contain the secondary part. The Program Register is loaded during I time which distinguishes an instruction which is only brought into the computer during I time and Data which is brought in during E time. The program register is loaded normally with an I6 (D2) pulse. It is reset the following I6 D1. During manual operation we have a special I0(D3) reset.

Note: See section in Instruction Overlap for special conditions of setting P. R.

214 Sense Indicator Register (SI) Systems 02.06.00.1-02.06.11.1

The Sense Indicator Register is a 36 position register labeled 0-35. Each position is a single trigger with a delayed output. The SI register is controlled completely by the program and is not used by the computer as a part of its arithmetic operations. It can be used as a set of switches which are set and tested by the program to check the progress or direction of the program. The SI register may also be used to store words or parts of words temporarily and, in this way, it is useful for altering and testing words. The only path for information into or out of the SI's is by way of the SR.

215 Shift Counter (SC) Systems 03.04.14.1-03.04.17.1

The Shift Counter is an 8 position count down counter labeled 10-17, each position consisting of a single trigger with a delayed output. The delay prevents interaction between groups when stepping the counter. The SC counts the number of shifts taken during a shift operation (MPY, DIV, ALS, ETC) and is also used as part of the decoder for operations that have a primary operation of 76.

Because the SC is a step down counter, a number must be gated into the counter. The time at which it is gated is controlled by the operation being performed. During a primary operation of 76 the XAD are gated to the SC during I time. During a Variable Length MPY-DIV the XAD are gated to the SC during E time and during a Convert instruction during L time. The SC is used during FLOATING POINT ALSO. While performing a FAD instruction it is used to line up our characteristics. At E11 time the characteristic difference between the SR and ACC which is computed in the Adders is gated to the SC to keep track of the number of shifts necessary to align the fractions. Adders 3-8 are gated to the SC 12-17.

During a MPY-DIV operation a number must be gated into the SC to keep track of the shifting but it does not come from another register. It is a constant which is used during these operations and will always be the same. In the case of a DIV operation 33_8 is set into the SC and for a MPY operation 43_8 is set to the SC during E time.

The SC is stepped every clock pulse providing a gate called SHIFT Gate is conditioned. See Figure an generation of Shift Gate and Figure for stepping of the SC.

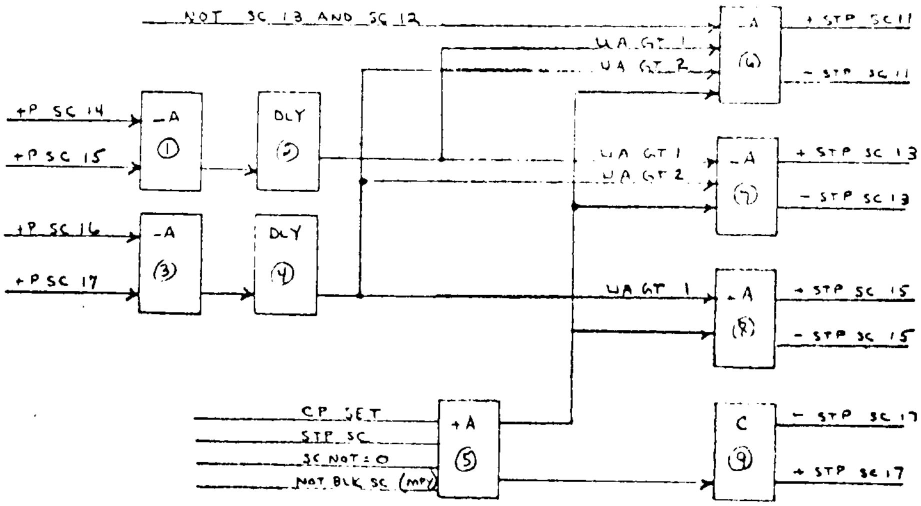
STEPPING THE SHIFT CTR

In Figure B we have shown only 4 positions of the SC for simplicity.

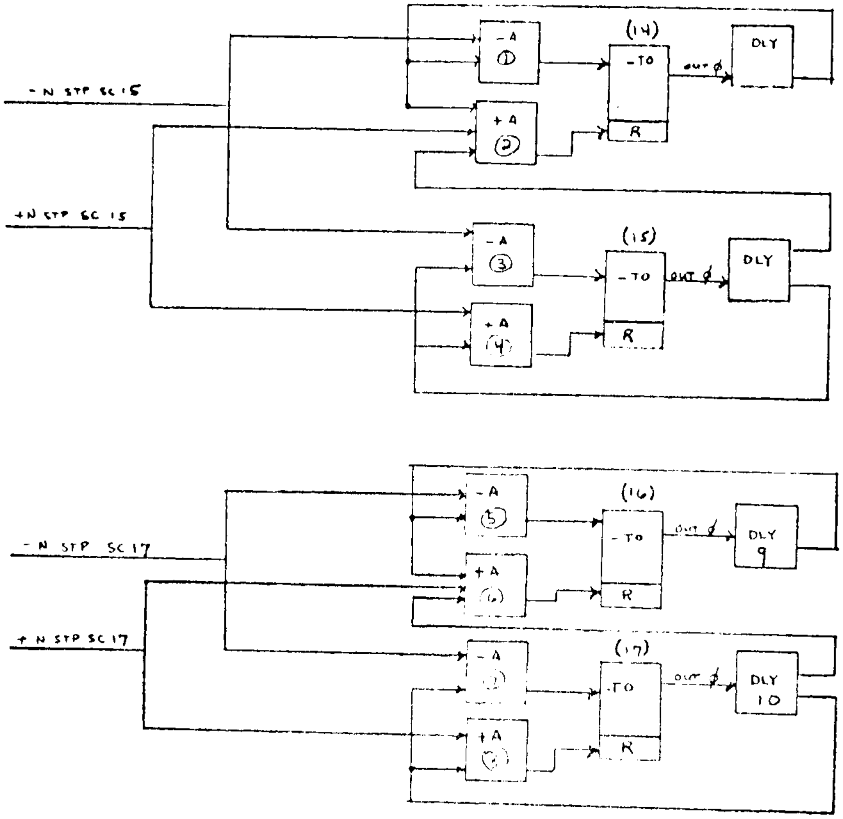
Let's assume that the SC is loaded with 17_8. Initially all the -T. O's are on in Figure B and all input AND's 1-8 are deconditioned until the step pulses come. Figure A shows the formation of the step pulses which uses a form of lookahead.

When the 1st step pulse comes the +A at 5 in Figure A will be conditioned and bring up the outputs of the C at 9 step SC 17. The +A at 8 in Figure B will now be conditioned resetting SC17. The out-of-phase output of the DLY at 10 will be minus preventing the reset of 16. The -A at 5 and -A at 7 can not be affected because the output of the DLY at 9 and 10 were plus when the step pulse came. The next step pulse will again condition the +A at 5 and bring up step SC 17 again. Notice that the Step SC 15 line cannot come up because it is conditioned by 16 and 17 being off. When the second step to 17 comes the +A at 6 (Figure B) and the -A at 7 will be conditioned due to 17 being off and 17 will be turned on and 16 turned off. The third step pulse will again bring up STP SC 17 which will reset 17 thru +A at 8 but cannot affect 16 because the +A at 6 is deconditioned by the output of the DLY at 10. Note use of delay here. The 4th step pulse will condition the +A at 5 Figure A again but this time the -A at 3 will also be conditioned bringing up the -A at 8. The purpose of the delay at 4 was to prevent the step pulse that stepped the counter to 14 from affecting the Stp SC 15 line.

The stepping process continues in this manner until the SC = 0. Notice that STP SC 13 cannot be conditioned until 14, 15, 16, 17 are off.

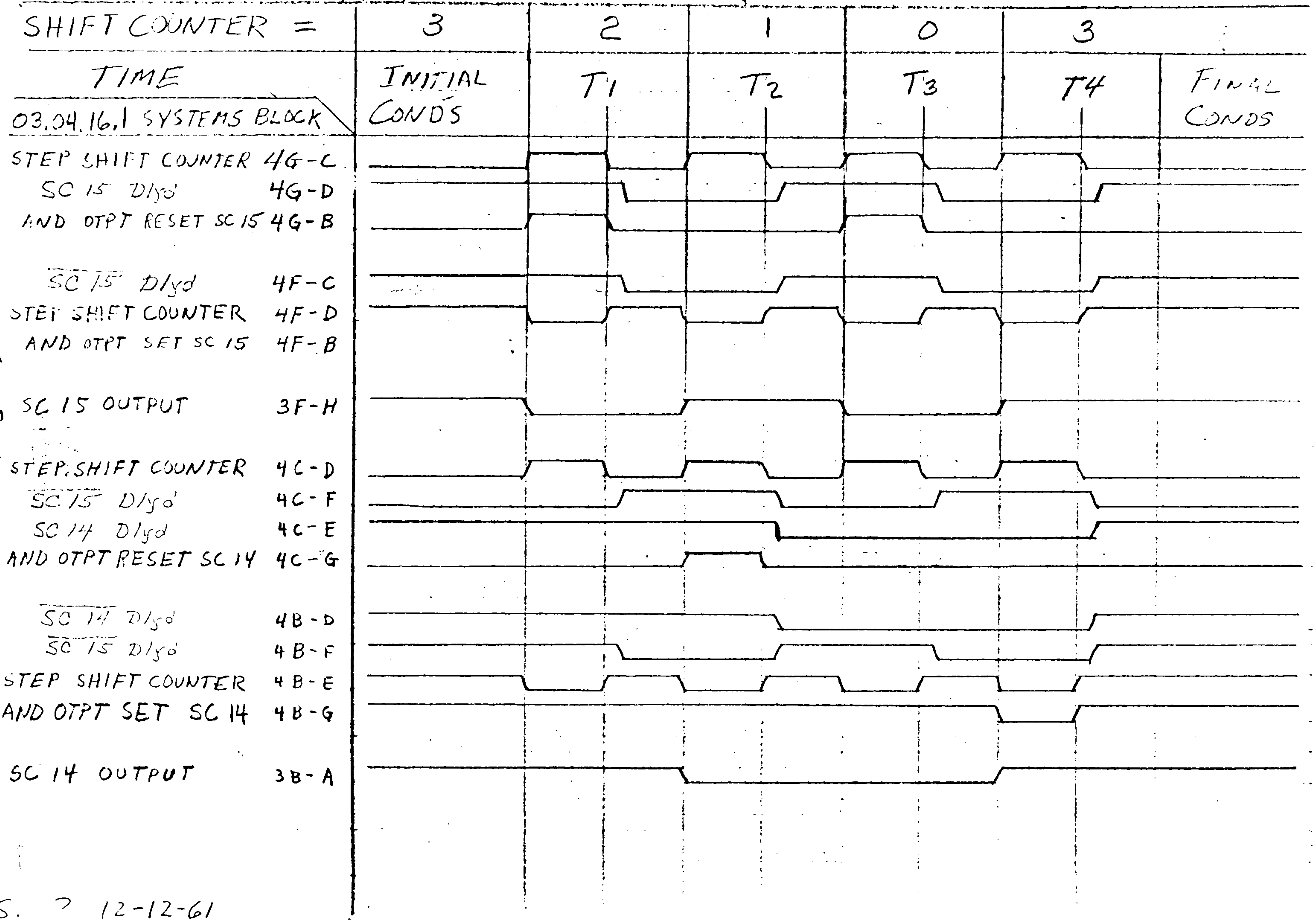

NOT SC 13 AND SC 12
UA GT 1
UA GT 2
+P SC 14
+P SC 15
-A
①
DLY
②
+P SC 16
+P SC 17
-A
③
DLY
④
-A
⑥
+STP SC 11
-STP SC 11
UA GT 1
UA GT 2
-A
⑦
+STP SC 13
-STP SC 13
UA GT 1
-A
⑧
+STP SC 15
-STP SC 15
CP SET
STP SC
SC NOT = 0
NOT BLK SC (MPY)
+A
⑤
C
⑨
-STP SC 17
+STP SC 17

(FIG A)


-N STP SC 15
+N STP SC 15
-A
①
+A
②
(14)
-TO
R
OUT ϕ
DLY
-A
③
+A
④
(15)
-TO
R
OUT ϕ
DLY
-N STP SC 17
+N STP SC 17
-A
⑤
+A
⑥
(16)
-TO
R
OUT ϕ
DLY
9
-A
⑦
+A
⑧
(17)
-TO
R
OUT ϕ
DLY
10

(FIG B)

SHIFT COUNTER CYCLE



2/5.4

S. ? 12-12-61

DOUBLE LATCHES
7094 ~~SHIFT CELLS~~

Shift cells used in the 7094 storage register and accumulator and multiplier-quotient registers have been especially designed to:

-- increase system reliability,

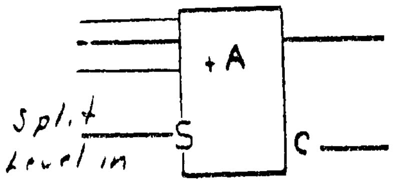
-- reduce component cost and space requirements, and

-- reduce the number of logic blocks required for the storage-shift function.

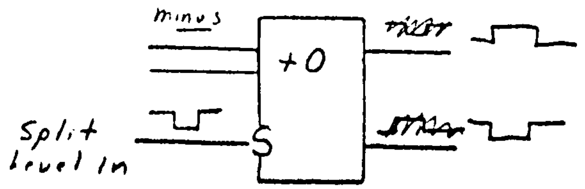
Accomplishment of these design goals resulted in shift cells affording simultaneous read-in and read-out; a double latch configuration in which inputs can be sensed prior to being "locked-in" and faster read-in and read-out as compared to the double trigger configuration previously used.

The 7094 shift cells use third and split level logic. A review of these different logic blocks is contained in TRANSISTOR COMPONENT CIRCUITS Form 223-6889-1. For this discussion it will suffice to recall the following:

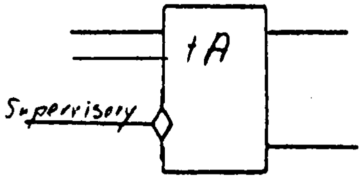
1. A split level AND circuit functions normally when the split level applied is up (active). The in-phase output is up and the out-of-phase output down when the level is down.



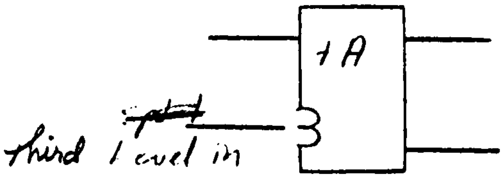
2. A split level OR circuit functions normally when the split level input is down. When the split level is up, the in-phase output is always down and the out-of-phase output is always up.



3. A third level AND circuit functions normally when the third (supervisory) level is up. When the supervisory level is down the logic block is deactivated and both the in-phase and out-of-phase outputs are down.



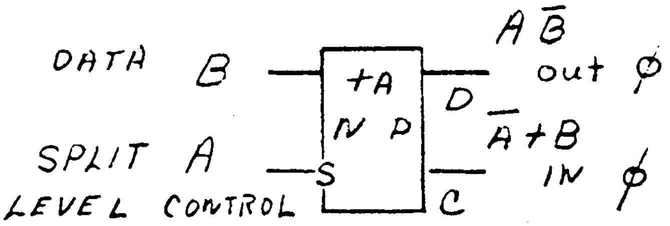
4. The symbol used for ~~distinction between~~ third and split level circuits is determined by location ~~circuit~~ of the coupling network, ~~loading,~~ i.e., external or internal. The symbology for a third level, positive AND circuit follows.



The shift cells function as temporary storage units and provide for transferring information from one register to the other, or to adjacent positions within a register. In the following example, Figure 217-1, a single shift cell of the storage register is illustrated. Note that inputs may come from the MQ, AC or AD. Outputs may be gated to the SB, AC or back around to the SR.

Split Level Logic Blocks

"And"

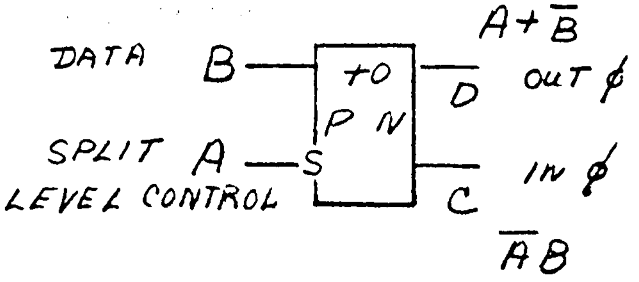


	T_1	T_2	T_3	T_4	
A	−	−	+	+	CONTROL
B	−	+	−	+	DATA
C	+	+	−	+	IN φ
D	−	−	+	−	OUT φ

CONTROL +, OUTPUTS FOLLOW DATA

CONTROL −, IN φ IS +, OUT φ −, DATA HAS NO EFFECT

"Or"



	T_1	T_2	T_3	T_4	
A	−	−	+	+	CONTROL
B	−	+	−	+	DATA
C	−	+	−	−	IN φ
D	+	−	+	+	OUT φ

CONTROL −, OUT PUTS FOLLOW DATA

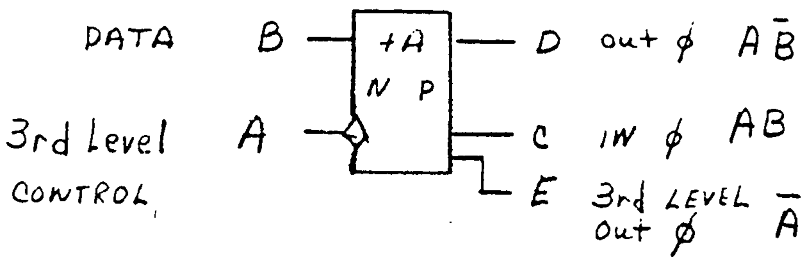
CONTROL + IN φ IS −, OUT φ +, DATA HAS NO EFFECT

φ = PHASE

MFS 5/12/[illegible]

Third Level Logic Blocks

"And"



	T_1	T_2	T_3	T_4	
A	−	−	+	+	CONTROL
B	−	+	−	+	DATA
C	−	−	−	+	IN φ
D	−	−	+	−	OUT φ
E	+	+	−	−	out φ 3rd

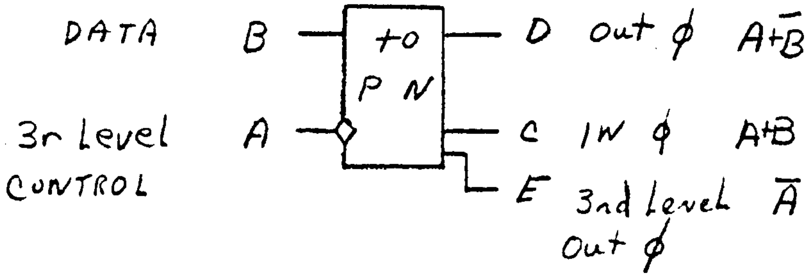
CONTROL +, OUTPUTS FOLLOW DATA

CONTROL −, IN φ AND OUT φ −, DATA HAS NO EFFECT

~~3rd LEVEL OUT IS THE OUT φ OF THE CONTROL~~

Pin E is diverted phase output (ie - unconditional out of phase from 3rd level input)

"Or"



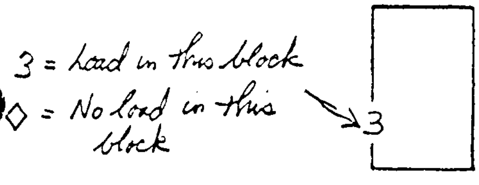
	T_1	T_2	T_3	T_4	
A	−	−	+	+	CONTROL
B	−	+	−	+	DATA
C	−	+	+	+	IN φ
D	+	−	+	+	out φ
E	+	+	−	−	out φ 3rd

CONTROL −, OUTPUTS FOLLOW DATA

CONTROL +, IN φ AND OUT φ +, DATA HAS NO EFFECT

~~3rd LEVEL OUT IS THE OUT φ OF THE CONTROL~~

Pin E is diverted phase output (ie - unconditional out of phase from 3rd level input)



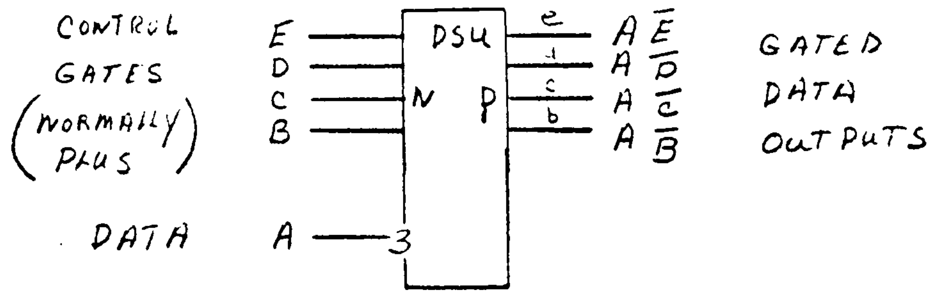
ALTERNATE LOGIC BLOCK

φ = PHASE

MFS 5/12/62

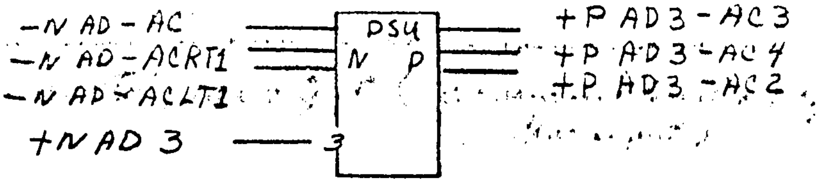
Distributor

3rd Level Logic



		T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
DATA	A	−	−	−	−	−	+	+	+	+	+
CONTROL GATES	B	+	−	+	+	+	+	−	+	+	+
	C	+	+	−	+	+	+	+	−	+	+
	D	+	+	+	−	+	+	+	+	−	+
	E	+	+	+	+	−	+	+	+	+	−
GATED DATA OUTPUTS	b	−	−	−	−	−	−	+	−	−	−
	c	−	−	−	−	−	−	−	+	−	−
	d	−	−	−	−	−	−	−	−	+	−
	e	−	−	−	−	−	−	−	−	−	+

ADDER 3 DIST.



NOTE!
RESTRICTIONS ONLY ONE CONTROL GATE DOWN AT A TIME

MFS
5/12/62

Read-In Sequence

Positive OR number 1 (OR-1) and Positive AND number 1 (AND-1) comprise the LTH logic block of the shift cell (so designated on the ALD pages). Some of the flow diagrams relating to the CPU show this as the first latch.

The second latch or register (REG) is made up of positive AND number 2 (AND-2). positive OR number 2 (OR-2) and positive AND number 3 (AND-3).

The terms first and second latch refer to the sequence of read-in to the shift cell. For the following discussion assume the shift cell has been storing a zero and a one is read in at time T1. Refer to the timing diagram, Figure 217-2, to establish the time relationships of the following events.

1. At T1, the input at X rises, indicating a 1 is being read in. This input has a duration of one clock pulse or 167 nanoseconds.

2. The split level input to OR-1 is down (AND-1 is disabled) so this logic block functions as a normal OR:

3. The in-phase output at pin Z rises and the output at pin V falls.

4. The pin G input to AND-2 is down. This circuit is disabled because the supervisory level is down.

5. At T1+ the set SR pulse is applied to the storage register. This ~~80~~ 72 nanosecond pulse is bracketed by the 167 nanosecond read-in pulse.

6. With the set pulse applied, supervisory levels are applied to AND-1 and AND-2, enabling them to function as normal AND circuits.

7. The input to pin M of AND-1 is up, and, because this is now functioning as a normal AND circuit, the output at pin F is up. Note that pin F is tied back to pin X, at the input of OR-1, forming a holding circuit. The input could now be removed without affecting the operation of the shift cell.

8. The pin H output of AND-1 is down and fed to pin K, the split level input of OR-1. OR-1 continues to function as a normal OR circuit, with the Z output ~~down~~ up and the V output ~~up~~ down. This sequence constitutes the first latch.

9. The pin V output of OR-1 is applied to pin G of AND-2. Because the set pulse is still up, this AND circuit functions normally and the output at pin C will be down, output at pin B will be up.

10. The pin B output is applied to the split level input of OR-2, causing the pin E output to go down and pin D output to go up.

11. The up level at pin D of OR-2 is applied to the ~~split~~ 3rd level input of AND-3. When the ~~split~~ 3rd level is up, the pin R output is down. This forms the holding circuit for the register.

12. At the end of the set pulse OR-2 functions normally and the down input at pin N (from AND-3) results in a down level at pin E and an up level at pin D. This sequence constitutes the second latch. The one may be read out of the register by pulsing any ~~or all~~ one of the three -N gate input lines to the register. With AND-3 functioning normally, these negative input pulses will result in up levels at the out-of-phase output. Conversely, when the register is holding a zero, supervisory input down, the register (AND-3) is cut off and the negative input pulses will result in negative output pulses -- zeros.

The time sequence is such it is possible to read into the first latch of the shift cell while reading out of the second latch.

7094 Storage Register Position

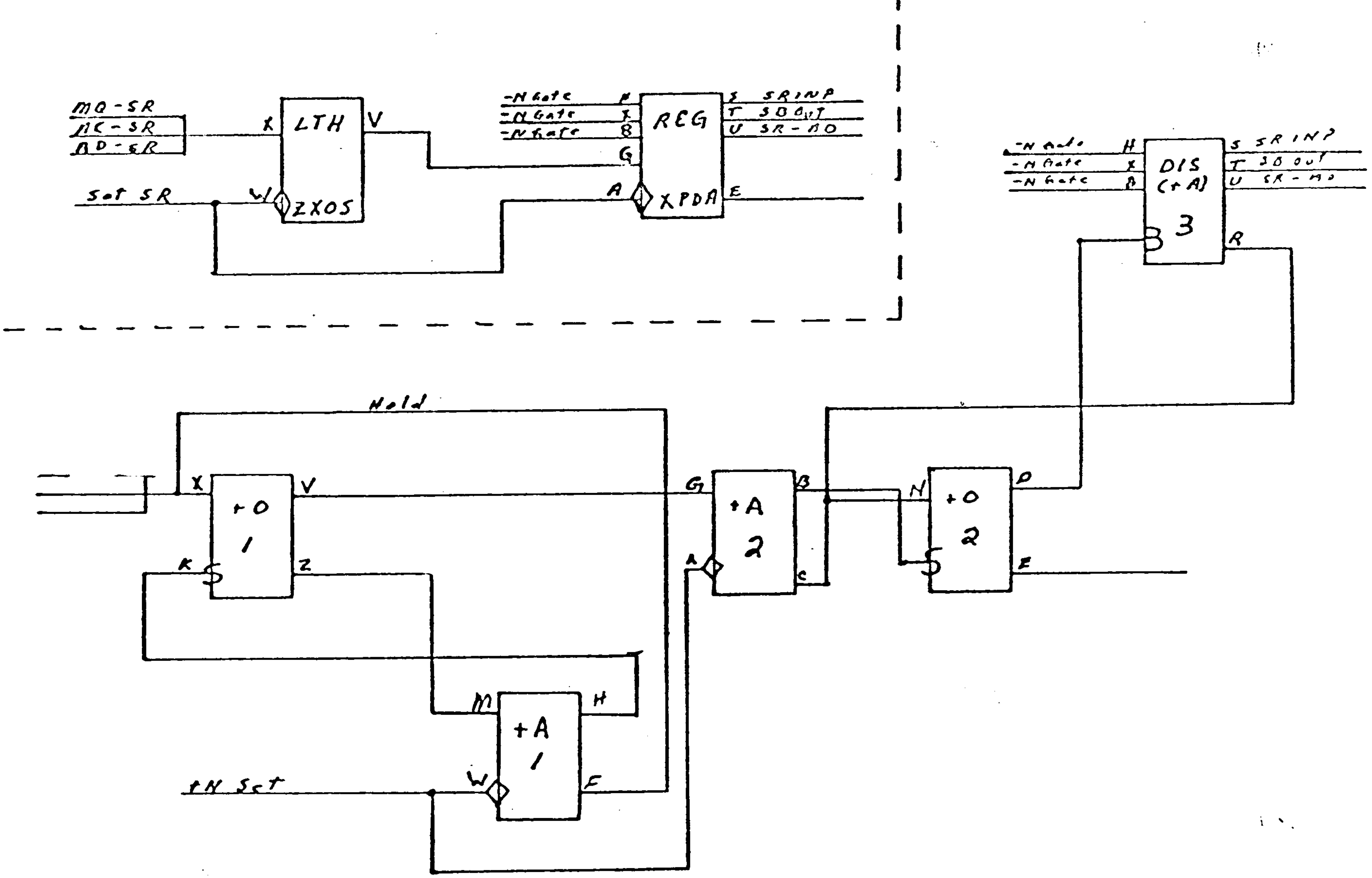
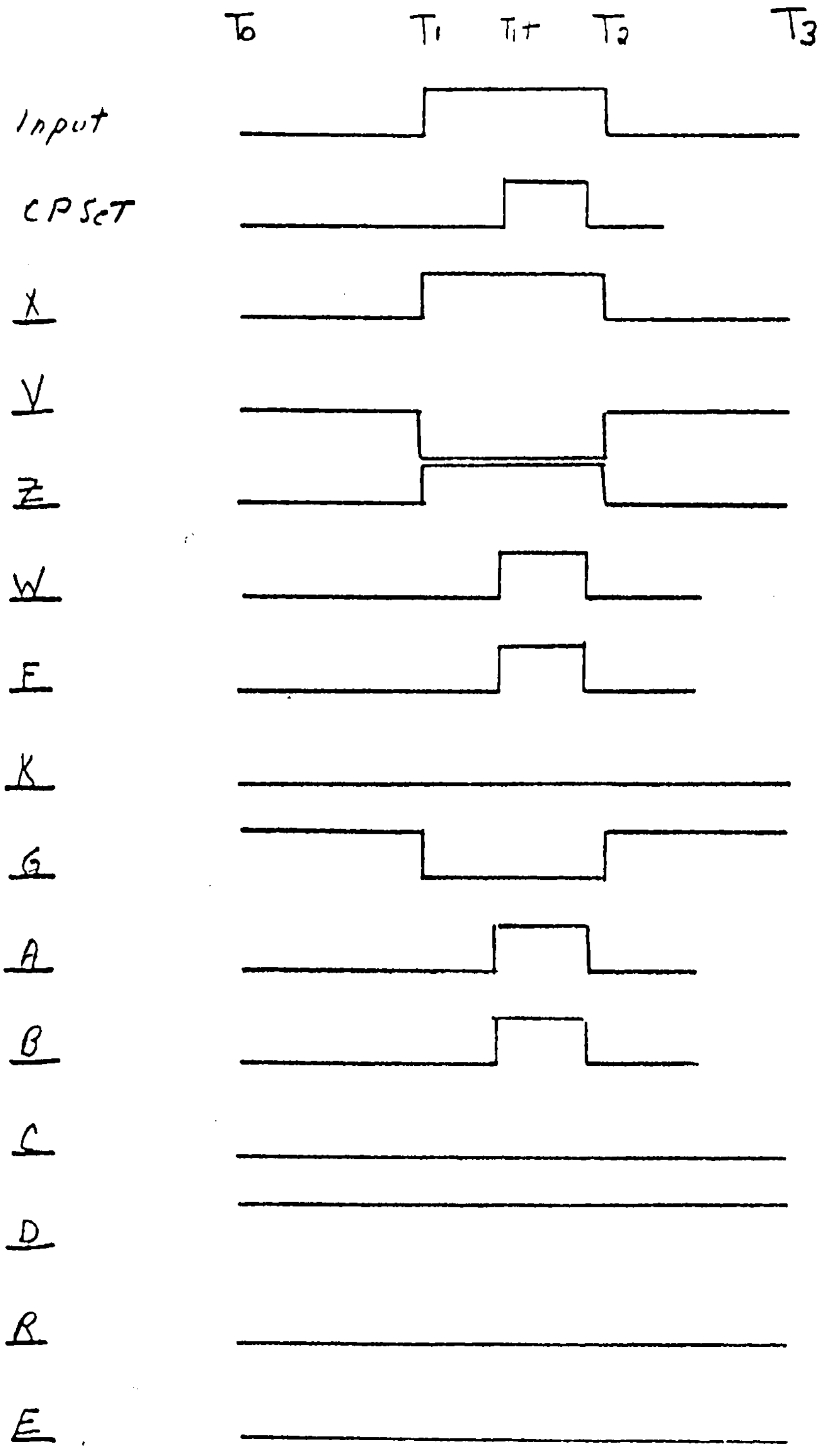


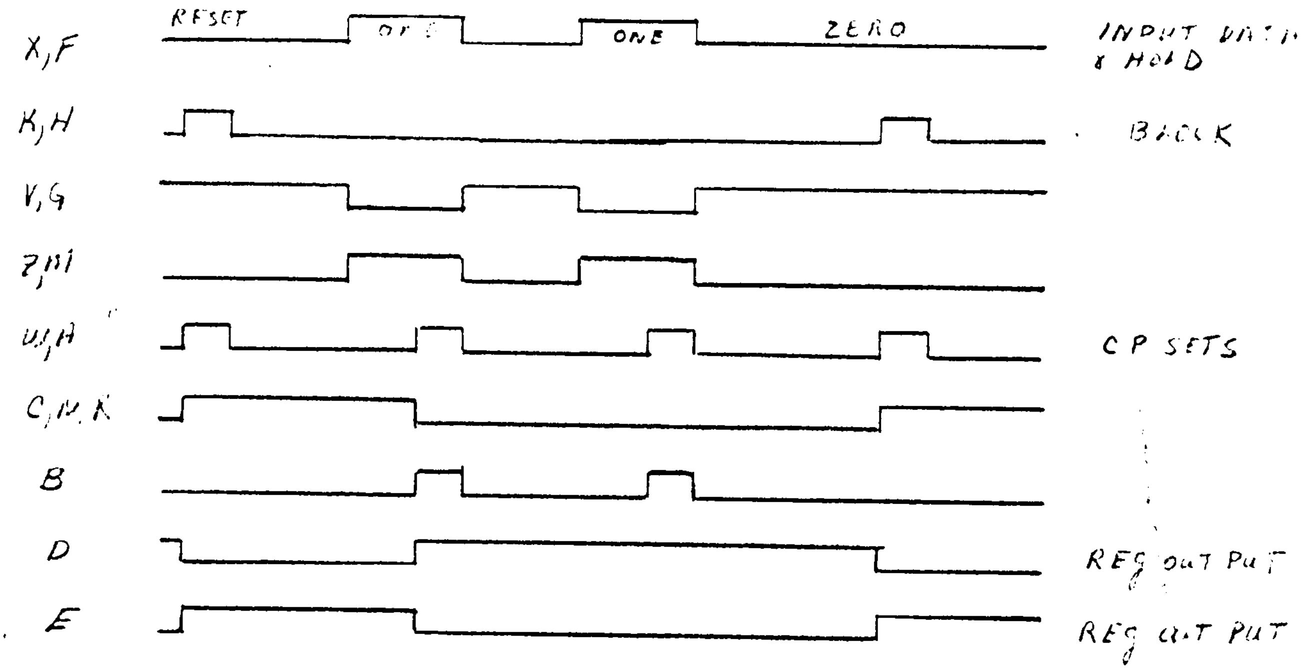
Figure 217-1



Note : Waveforms shown are idealized.
Timing shown is approximate.

Figure 217-2 . Sequence Chart For Storage Register Position Read-In

7094 Storage Register Timing Sequence (Figure 217-1 page 217.5)



OUTPUT E minus when register contains a ONE

P 217.6A

MFS
5/13/62

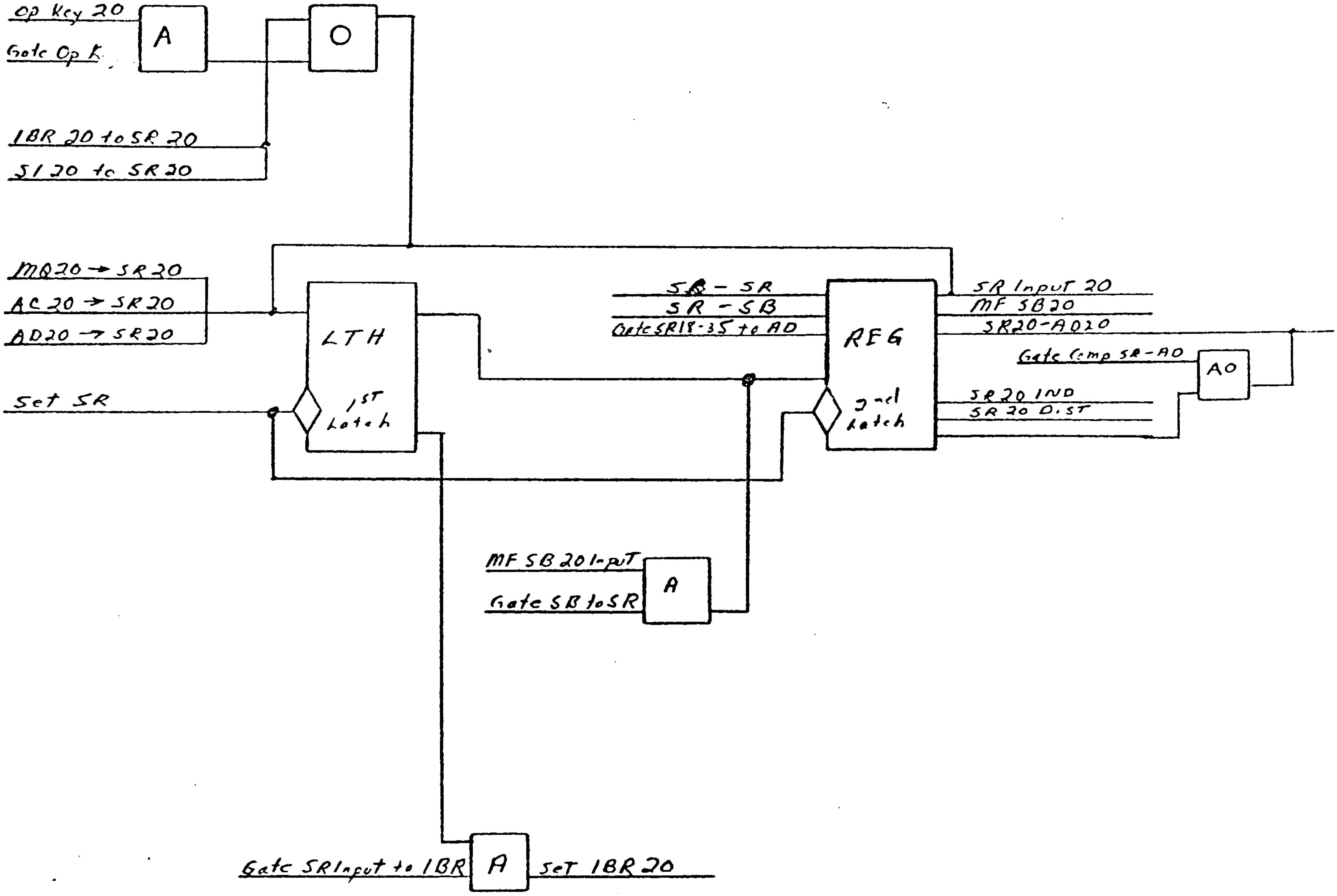


Figure 217-3 STORAGE REGISTER BIT 20

TAG REGISTER (Systems 03.05.20.1)

The Tag Register is a three position trigger register numbered 18, 19, and 20. It is fed directly from the Multiplexor Storage Buses and is used to hold the tag portion of an instruction during the instruction execution. The Tag Registers are used to inform the system which Index Register (XR) or registers are going to be operated upon. The outputs of the Tag Register may be used singly or in combination to specify the XR needed. Index Register tag coding is:

Index	Tag	Column	Index Register Name
18	19	20	
0	0	1	A (XR1)
0	1	0	B (XR2)
0	1	1	D (XR3)
1	0	0	C (XR4)
1	0	1	E (XR5)
1	1	0	F (XR6)
1	1	1	G (XR7)

The seven index registers use all the combinations of these tag bits for individual addresses. Therefore, the presence of more than one tag bit does not mean the OR'ing together of the addressed XR contents.

The 7094, however, does provide compatibility with previous systems having only 3 XR's. The 7094 is normally operating in the three XR mode; but by giving the instruction LMTM (-0760---16) the computer is placed in seven XR mode. It also turns off the Multiple Tag MODE Indicator on the CPU Console. In order to return to normal 3 Index Register mode the instruction EMTM is given.

220 Floating Point Tally Counter (Systems 02.10.21.1-02.10.22.1)

The Tally Counter is a three stage counter whose output is ANDed with L time to direct various phases of Floating Multiply and Divide. For a more detailed description of the Tally Counter and its uses see Floating Point Section.

300

7094 TIMING

301 Timing Function

All computer functions are directly related to a master computer clock. This master clock, located in the multiplexor, establishes a basic rate of performance of two microseconds for each machine cycle.

Machine cycles are determined by gating the output pulses of the master clock, twelve for each two microseconds, to allow certain distinct functions to be performed during the cycle. These permissive circuits are the logic building blocks of the computer, AND and OR circuits and triggers. By gating the master clock pulses under specific circumstances and sequences the computer operates in E , I, L or B time, or, the computer is performing an I, E, L or B cycle.

Computer logic limits performance during any particular cycle to the following :

1. During an I cycle the computer normally ~~must~~ makes a reference to core storage to bring out an instruction.

2. During an E cycle the computer must be executing an instruction requiring a~~n operand from~~ core storage. reference.

3. During an L cycle the computer must be performing a logic or arithmetic function, not requiring a reference to core storage.

4. During a B cycle the computer must be accepting or delivering information to the input-output equipment.

From these definitions it can be seen the abbreviations refer to computer activity, thus, I for Instruction cycle, E for execute cycle, L for Logic cycle, and B for Buffer cycle.

Change

The computer's cyclic sequence is fixed, always beginning with an instruction cycle and progressing to an execute, ~~or~~ logic or instruction cycle, determined by the instruction. The computer will go to B time as the need to use I/O equipment arises.

Although the sequence is fixed, the number of cycles allowed, exclusive of instruction, is not. The number of cycles required for each function is determined by the number of steps to be performed before the answer is complete. The number of cycles for each instruction can vary from as many as 19 to as few as one.

A clear and add (CLA) instruction requires two cycles, an I and an E. During the instruction cycle the computer will locate the instruction in core storage and bring it into the program register for decoding. At the end of two microseconds the computer will begin the E cycle, bring the opearnd out of core storage and into the accumulator. After the two microseconds of E time the computer will go back to I time and locate its next instruction in core storage.

302 Master Clock Pulses

The clock pulses used throughout the computer are developed in the multiplexor. The output of a three megacycle oscillator is cathode-coupled to the clock pulse-shaping network. All negative signal excursions are lost because of cathode coupling so the output is a string of positive pulses at the three megacycle rate as shown in Figure 301. These pulses are applied to converter blocks to become set drive pulses and N- and P-level clock drive pulses. The set pulses will be discussed in a later section.

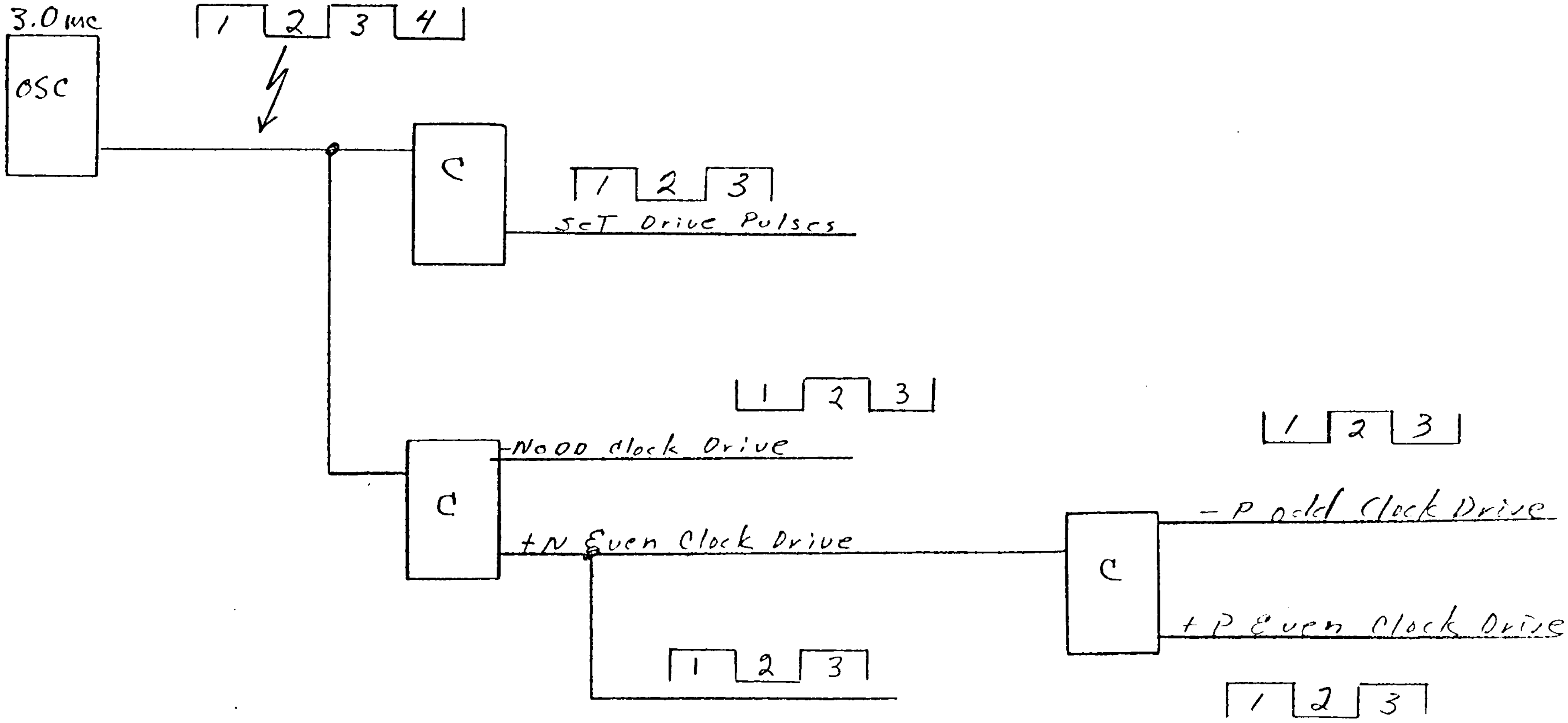


Figure 301. 3-Megacycle Oscillator Output and Converter Network

The odd and even clock drive pulses are applied to a 12-stage ring of triggers making up the actual multiplexor clock. Two triggers are shown in Figure 302, with their inputs and outputs. The clock is started (trigger 0) by turning on the trigger with a start clock pulse. The A0(Dl) * clock pulse output is obtained by gating the last half of the zero clock trigger with the negative portion of the even clock drive pulse. Rise of the A0(Dl) pulse turns on clock trigger l. The Al(Dl) pulse output is obtained by gating the last half of clock trigger l pulse with the negative portion of the odd clock drive pulse.

Rise of the Al(Dl) pulse turns on clock trigger 2. Clock trigger 2 produces the A2(Dl) clock pulse and also turns off clock trigger 0. This sequence continues through clock trigger ll. Rise of the All(Dl) pulse turns clock trigger 0 back on and the sequence repeats. The following chart shows the controls significant to each stage of the ring.

Clock	Turned				Gated
Tgr	on by	Turned off by	Duration	Output	Output
0	All(Dl)	Clock tgr 2	All(D2)	All(D2)	A0(Dl)
1	A0(Dl)	Clock tgr 3	A0(D2)	A0(D2)	Al(Dl)
2	Al(Dl)	Clock tgr 4	Al(D2)	Al(D2)	A2(Dl)
3	A2(Dl)	Clock tgr 5	A2(D2)	A2(D2)	A3(Dl)
4	A3(Dl)	Clock tgr 6	A3(D2)	A3(D2)	A4(Dl)
5	A4(Dl)	Clock tgr 7	A4(D2)	A4(D2)	A5(Dl)
6	A5(Dl)	Clock tgr 8	A5(D2)	A5(D2)	A6(Dl)
7	A6(Dl)	Clock tgr 9	A6(D2)	A6(D2)	A7(Dl)
8	A7(Dl)	Clock tgr 10	A7(D2)	A7(D2)	A8(Dl)
9	A8(Dl)	Clock tgr ll	A8(D2)	A8(D2)	A9(Dl)
10	A9(Dl)	Clock tgr 0	A9(D2)	A9(D2)	Al0(Dl)
ll	Al0(Dl)	Clock tgr l	Al0(D2)	Al0(D2)	All(Dl)

Note that the pulse width of each clock trigger is twice that of an individual clock pulse. This slower switching of the clock triggers provides increased reliability in clock operation.

* the Dl in parentheses refers to pulse duration in terms of pulse width.

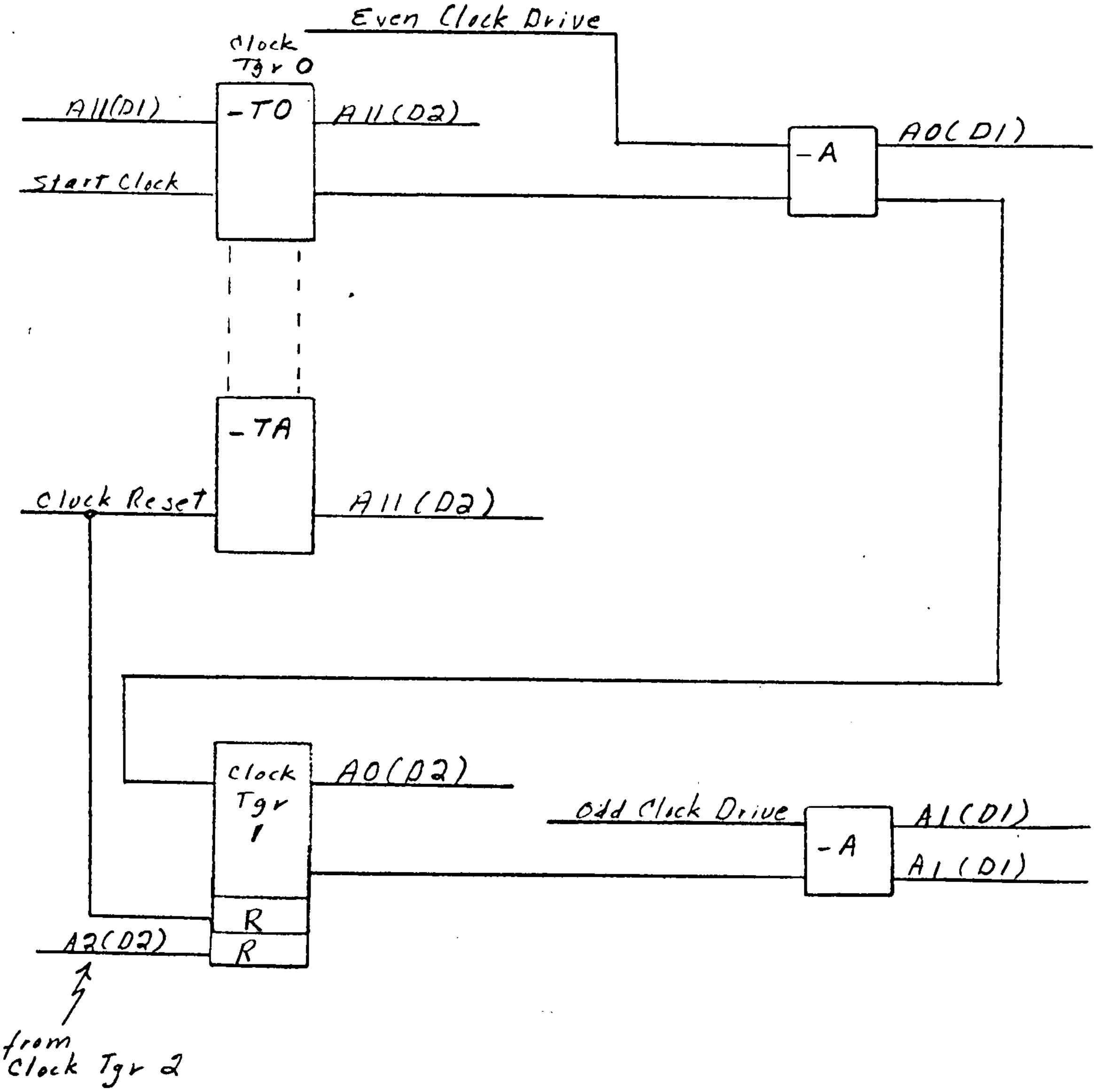


Figure 302. Triggers 0 and 1 of Multiplexor Clock

A timing sequence of the clock pulses is shown in Figure 303. These clock pulses are distributed to the computer and to the data channels. Because of inherent delays in logic blocks and cable transmission, clock pulses to the computer arrive about one clock pulse late. For this reason, computer clock pulses are labeled one pulse higher than those in the multiplexor. Therefore, an A0(D1) pulse going to the computer would be labeled A1(D1). This pulse leaves the multiplexor at A0 time but by the time it arrives at the computer the A1 pulse is rising in the multiplexor. A1 pulses in the computer and in the multiplexor are now in coincidence, although developed from different clock triggers. This provides continuity in the timing relationships between the computer and the multiplexor.

Although the A0(D1) through A11(D1) are the prime outputs of the multiplexor clock the D2 pulses are also distributed and used throughout the computer and the data channel.

In the computer these pulses are gated during cyclic operation and then labeled I6(D1) or E5(D1), depending on the particular cycle of operation. Whenever an A pulse is encountered in studying the computer, for example A6, it means this pulse is used directly off the clock, independent of the cycle of operation the computer is in. This pulse will always occur at 6 time and will always be available. Later on it will be shown it is possible to force the computer to leave one time for another, as, for example, to go from the middle of I time, I6, to E time, E7 and continue the cycle in E time. It is imperative there be A pulses available for gating or other logic operations during such transitional periods.

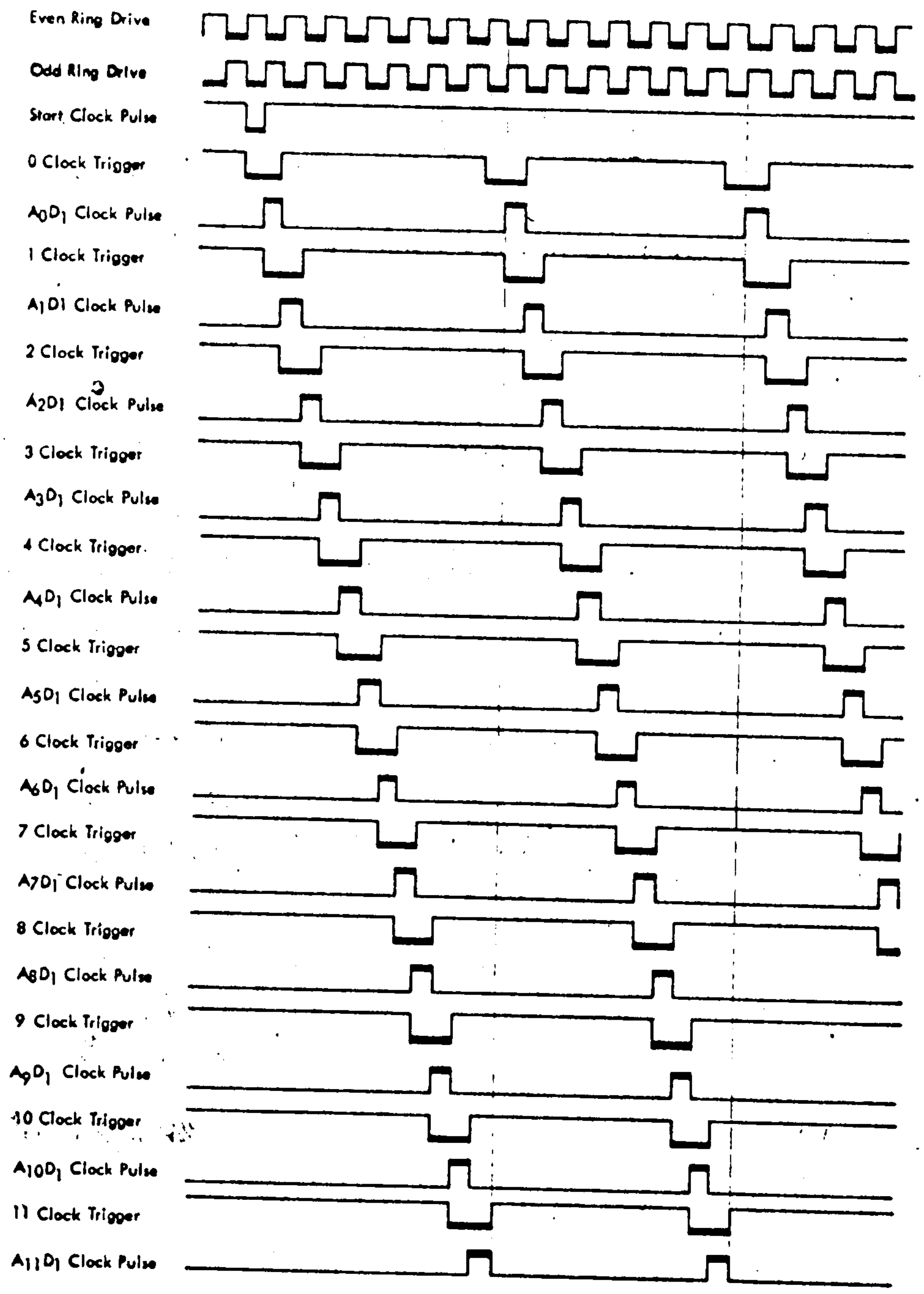


Figure 303 MULTIPLEXOR CLOCK

3Ø3 Computer Set Pulses

Computer set pulses (CP set pulses), developed off themaster oscillator, are used in the computer to set triggers and latches; as supervisory inputs to third level logic circuits and to control much of the gating and shifting within the computer. Width and timing of the CP set pulse, as related to the clock pulses, are extremely important to successful machine operation. Both of these factors are variable and will be set for each installation. Therefore, the timing discussion presented here can only be approximate.

Development of the CP setpulses is shown in Figure 304. The numerals refer to waveforms on Figure 305, indicating this waveform will be found at the numbered location on the functional drawing. Figure 305 also shows how the final CP set pulse is derived by inverting and delaying the clock pulses and then comparing them through AND circuits, as noted on Figure 304. The result is a string of positive pulses, occurring towards the end of the clock pulse and completely bracketed by it. At this point the CP set pulse is about 80 nanoseconds wide. Figures 2 and 2 show how the CP set pulse is applied to one stageof the storage register and its timing relationship to the read-in clock pulse (information).

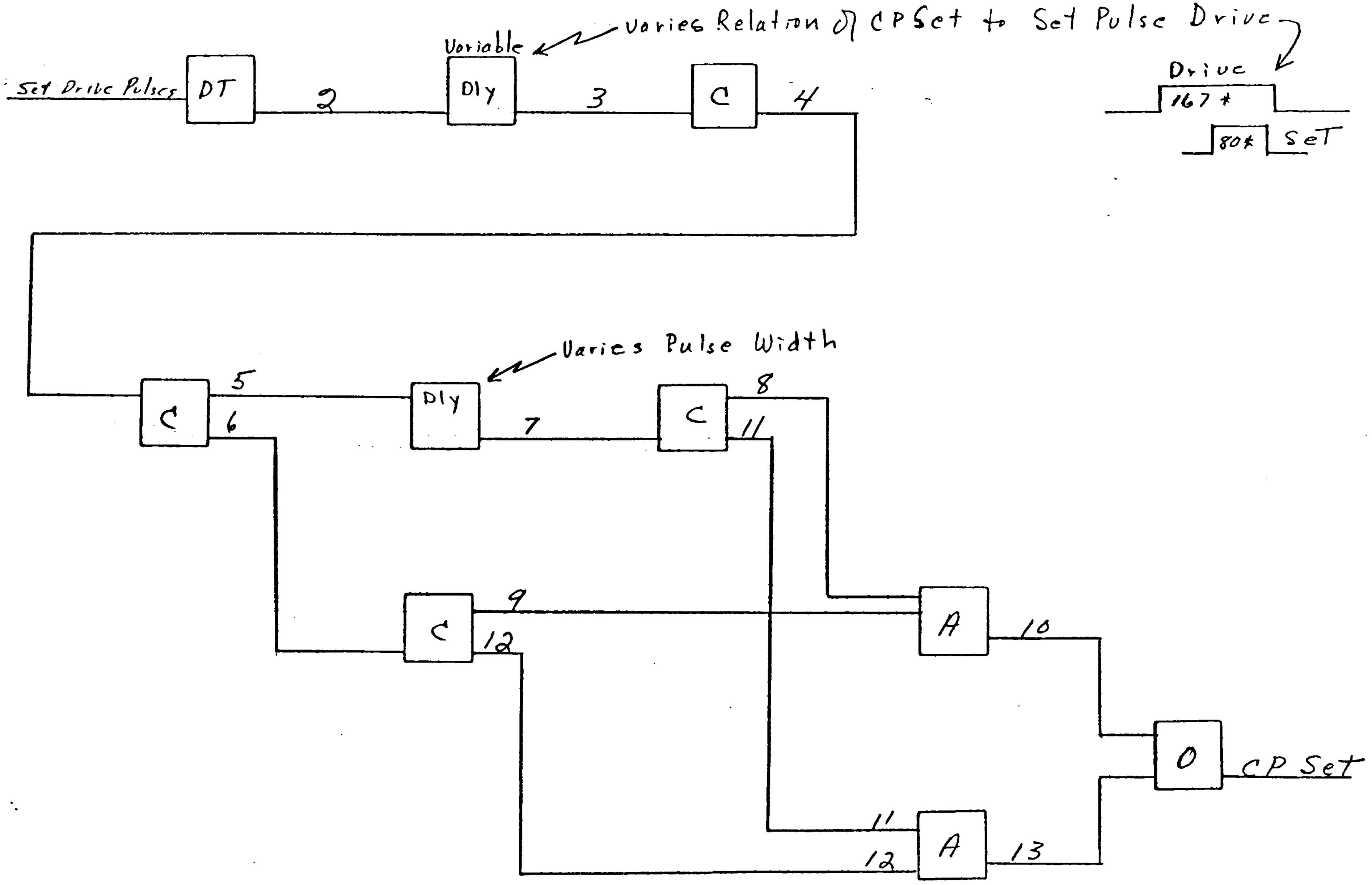


Figure 304. Development of CP Set Pulses

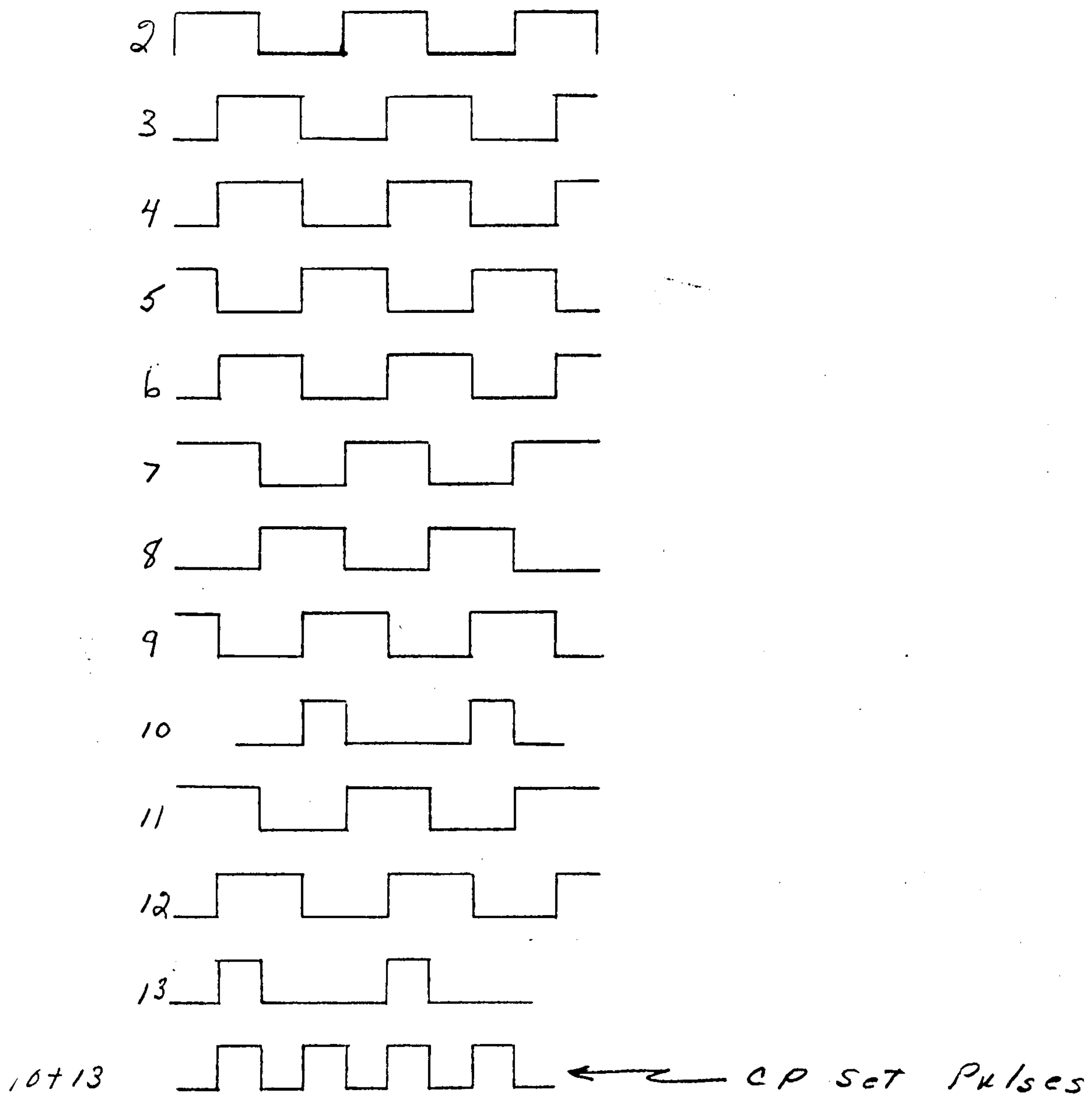


Figure 305. Development of CP Set Pulses from Clock Drive Pulses -- Shown Idealized

3Ø4 Timing Cycles

The four cycles of operation used by the computer indlucde:

a. Instruction (I)
b. Execution (E)
c. Logic (L)
d. Buffer (B)

Instruction Cycle -- during an I cycle the computer performs the following:

a. Locates and obtains the instruction from core storage.

b. Establishes the execution control cúrcuits for the instruction.

c. Locates the operand, if any, in core storage as specified by the instruction's address field.

Execution Cycle -- during an E cycle the computer ~~will~~ makesreference to core storage under the following conditions :

a. All instructions requiring an operand must have an E cycle following the I cycle.

b. Indirect addressing of an instruction always requires an extra E cycle. An instruction that would normally go from I to E for execution will go to I, E (for IA), and again to E if it is indirectly addressed.

c. Instructions with coding from 0200 to 0667 have an E cycle immediately following the I cycle. The remaining instructions usually go from I to L.

Logic Cycle -- an L cycle does not require a reference to core storage. Many instructions use both E and L cycles, when information is required from core storage and the instruction cannot be completed ~~[illegible]~~ during an E cycle. Those instructions not requiring a reference to core storage use only I or I and L cycles.

Buffer Cycle -- the computer goes to B time whenever it is necessary for the data channels to get information from or put information into core storage. Such information may consist of data or data channel commands. Because of the nature of input-output devices, all demands for a B cycle must take precedence over an instruction being performed in the computer. Otherwise, information coming from or going to a data channel might not be transmitted in time. A request for B time is honored during the cycle in which it is received. If the following cycle is an L cycle, the computer will share B and L time in simultaneous operation. If the following cycle is an I or E cycle, computer operation is interrupted and B time supplied to the channel requesting it.

Data processing always begins with an I cycle, the computer must have an instruction to work with. The nature of the instruction will determine subsequent operating cycles, i. e,, whether they are E or L and how many cycles are necessary to complete the operation. When an operation has been completed, the computer starts again with an I cycle. B cycle demands are honored at any time.

The sequence of cycles is determined by logical gating circuits. For example, when the master I time trigger is on,the master clock pulses are gated as I pulses, providing there is not a B cycle demand, and the normal instruction cycle sequence takes place. Near the end of I time an E cycle request may be initiated. Should this oocur, I time will fall at I11 and E time will start at E 11 and continue for 12 pulses.

Master I Time

Note : In this and following discussions it is important to bear in mind that although a trigger may be on, the output may not be gated to control the machine cycle.

Figure 306 illustrates the master I time trigger. Notice that the trigger is turned on when two conditions exist :

a. There is a "go to I time" signal.

b. It is A11 time.

The trigger called out by the double asterisk shows an end operation trigger is necessaryto get the "go to I time" signal. As shown in Figure 307, the end operation trigger is turned on at A10 when the computer has completed a full instruction. In other words, if the computer is completing an instruction during an E cycle, the end operation trigger will be turned on at E10. Notice the end operation trigger can be turned on during I, E or L time.

Referring back to Figure 306, notice that the AND circuit controlling the output of the master I time trigger can be disabled at any time by a B cycle interrupt (demand).

Assume the computer is completing an instruction and the end operation trigger is turned on at 10 time. This will bring up the "go to I time" signal (Figure 306) and on the next clock pulse (A11) the master I time trigger will be turned on. The following sequence of events takes place during I time (refer to Figure 308) :

1. The program counter was stepped during the preceding instruction cycle and at A10 this new instruction address was sent to the memory address register (MAR). By I 6 the instruction has

Master I Time (Systems 08.00.18.1)
+N

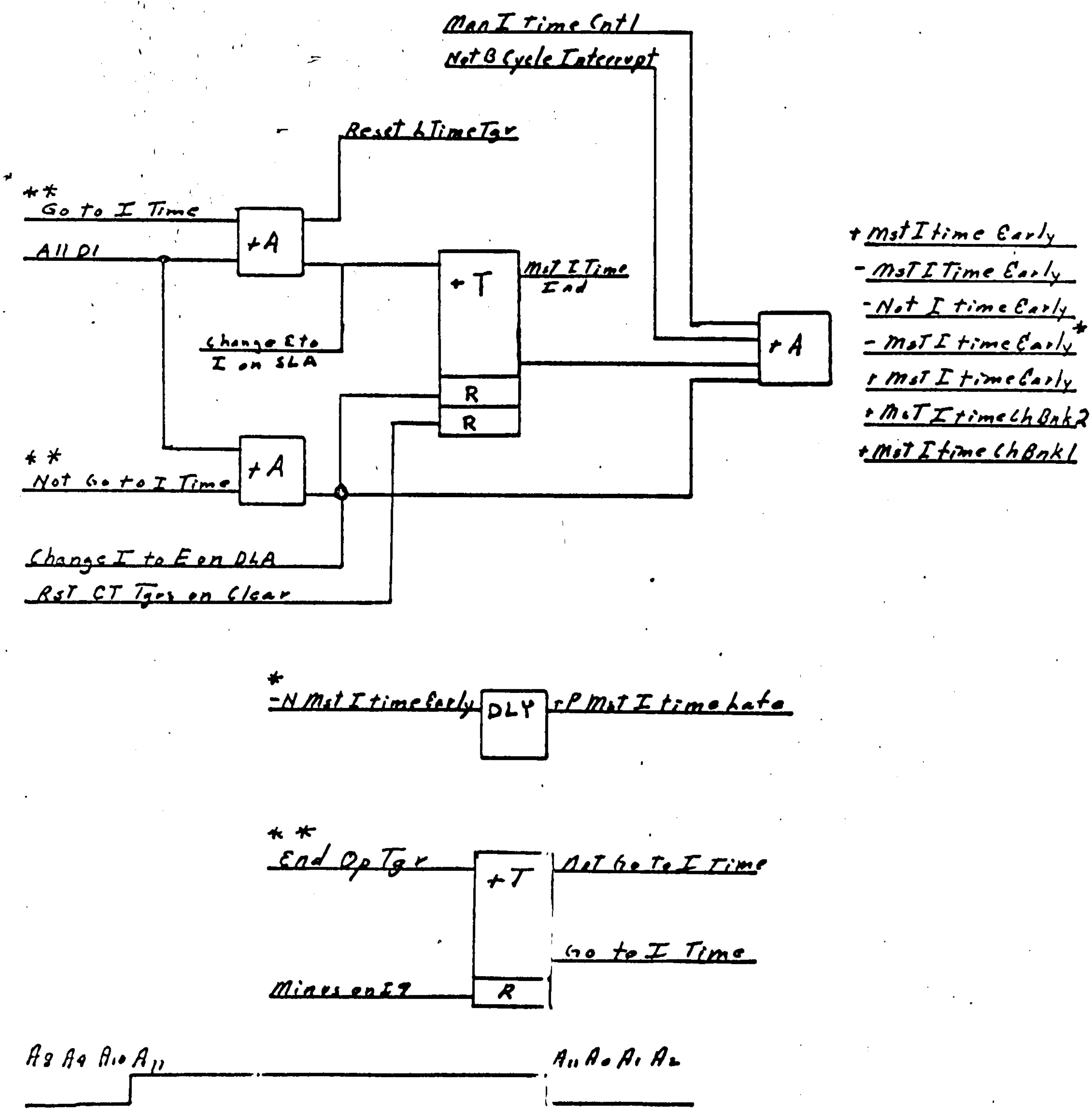


Figure 306. Master I Ti:: e Trigger

End Operation Trigger (Systems 08.08.09.1)
+ P

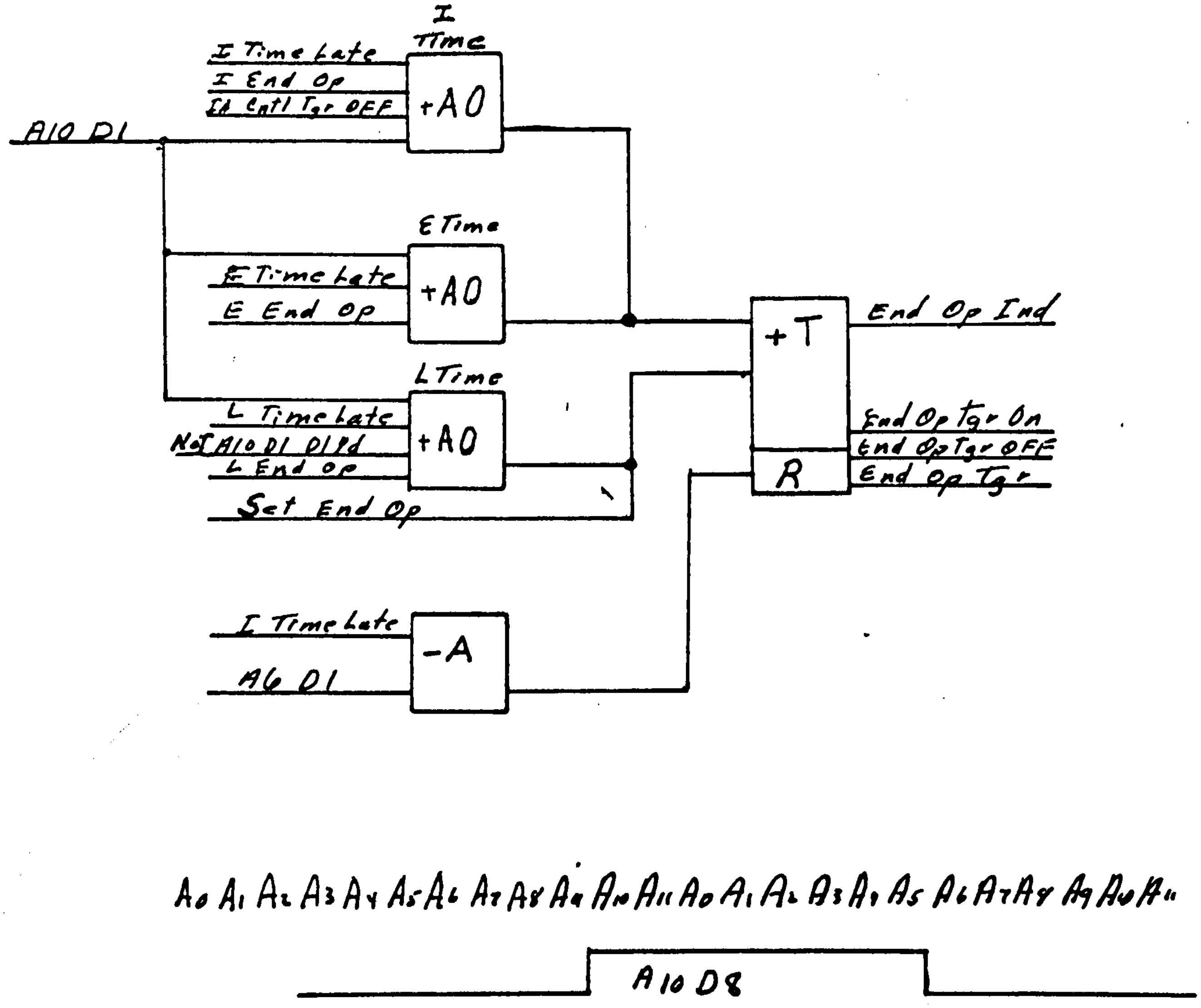
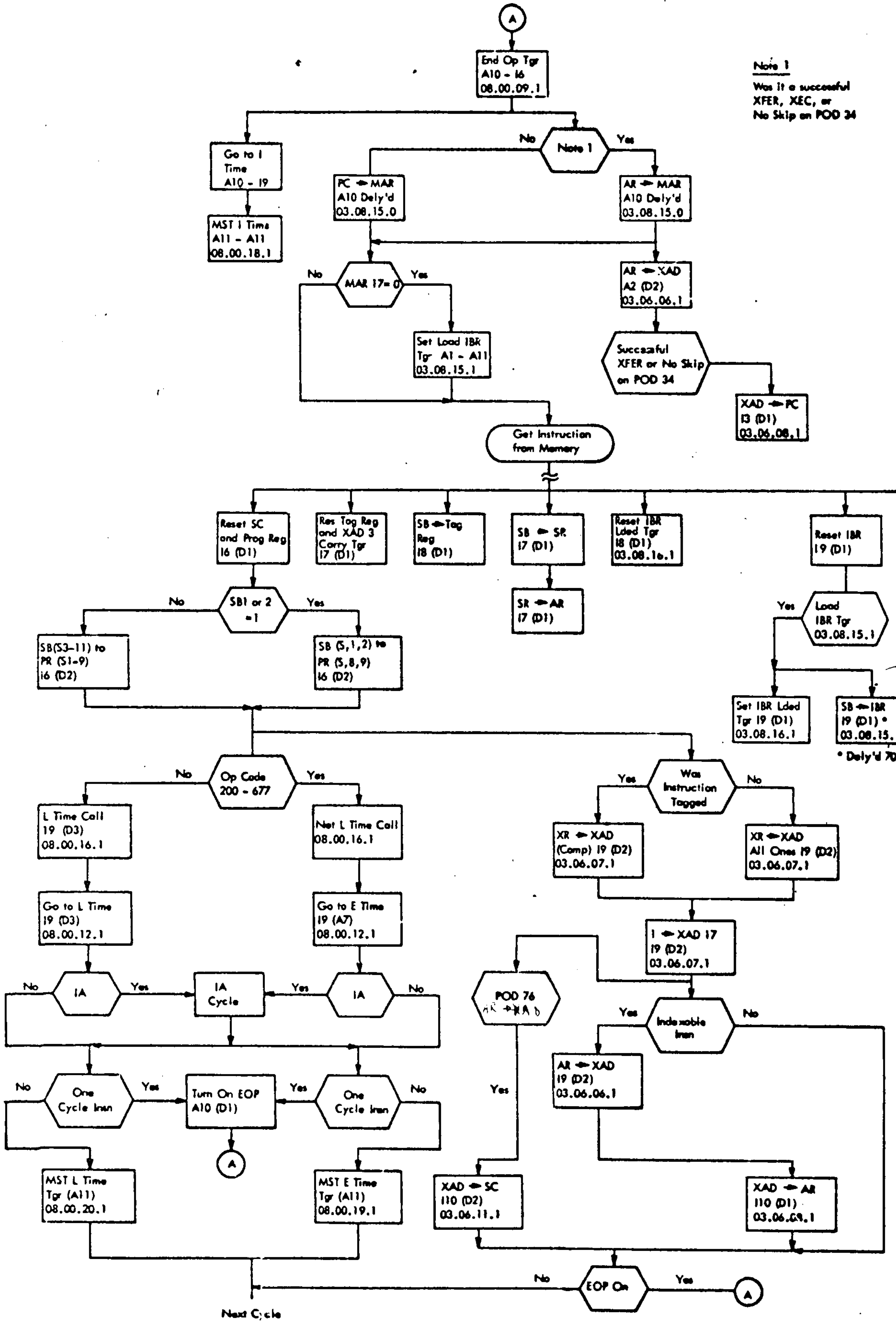


Figure 307 End Operation Trigger

INSTRUCTION CYCLE


A
End Op Tgr A10 - I6 08.00.09.1
Go to I Time A10 - I9
MST I Time A11 - A11 08.00.18.1
No
Note 1
Yes
PC ← MAR A10 Dely'd 03.08.15.0
AR ← MAR A10 Dely'd 03.08.15.0
No
MAR 17= 0
Yes
Set Load IBR Tgr A1 - A11 03.08.15.1
AR ← XAD A2 (D2) 03.06.06.1
Successful XFER or No Skip on POD 34
XAD ← PC I3 (D1) 03.06.08.1
Get Instruction from Memory
Reset SC and Prog Reg I6 (D1)
Res Tag Reg and XAD 3 Carry Tgr I7 (D1)
SB ← Tag Reg I8 (D1)
SB ← SR I7 (D1)
SR ← AR I7 (D1)
Reset IBR Lded Tgr I8 (D1) 03.08.16.1
Reset IBR I9 (D1)
PC ← XAD Hot 1 to 17 I7 (D2)
XAD to PC I8 (D1)
No
SB1 or 2 = 1
Yes
SB(S3-11) to PR (S1-9) I6 (D2)
SB (S,1,2) to PR (S,8,9) I6 (D2)
Yes
Load IBR Tgr 03.08.15.1
Set IBR Lded Tgr I9 (D1) 03.08.16.1
SB ← IBR I9 (D1) * 03.08.15.1
* Dely'd 70 *
No
Op Code 200 - 677
Yes
L Time Call I9 (D3) 08.00.16.1
Not L Time Call 08.00.16.1
Go to L Time I9 (D3) 08.00.12.1
Go to E Time I9 (A7) 08.00.12.1
No
IA
Yes
IA Cycle
Yes
IA
No
No
One Cycle Insn
Yes
Turn On EOP A10 (D1)
Yes
One Cycle Insn
No
A
MST L Time Tgr (A11) 08.00.20.1
MST E Time Tgr (A11) 08.00.19.1
Yes
Was Instruction Tagged
No
XR ← XAD (Comp) I9 (D2) 03.06.07.1
XR ← XAD All Ones I9 (D2) 03.06.07.1
1 ← XAD 17 I9 (D2) 03.06.07.1
POD 76
AR → XAD
Yes
Yes
Indexable Insn
No
AR ← XAD I9 (D2) 03.06.06.1
XAD ← SC I10 (D2) 03.06.11.1
XAD ← AR I10 (D1) 03.06.09.1
No
EOP On
Yes
A
Next Cycle

Note 1
Was it a successful XFER, XEC, or No Skip on POD 34

been located and brought to the output of the multiplexor (multiplexor storage bus input on the system flow diagram).

2. Also at I6(D1) the information in bits S,1-11 is gated to the program register for decoding. Decoding will bring up the necessary control lines to perform the function specified. These control lines will remain up until the next instruction is brought in and decoded.

3. At I7(D1) the incoming information is gated from the storage bus input to the storage bus and into the storage register. Bits 21-35 continue through the storage register to the address register. These bits contain the address of the operand needed to complete the instruction.

4. At I9(D2) the contents of the address register have been gated through the index adders for address modification, if any is specified. By the end of this two-pulse period the modified address is in the address register, ready to be gated to the multiplexor address switch. When this address is received in core storage the information at the address will be brought to the computer.

5. By I9 the information in the program register has been decoded and the appropriate control lines brought up to perform the specified function. The computer, therefore, knows whether an E or L cycle is needed to complete the instruction or whether this is a one cycle instruction and I time will be called for again.

6. At I9 an I, L or E time call is initiated which determines the next machine cycle. Figure 309 shows the conditions necessary to initiate E and L time calls. Figure 310 shows how these set the "go

E and L Time Call (Systems 08.00.16.1)

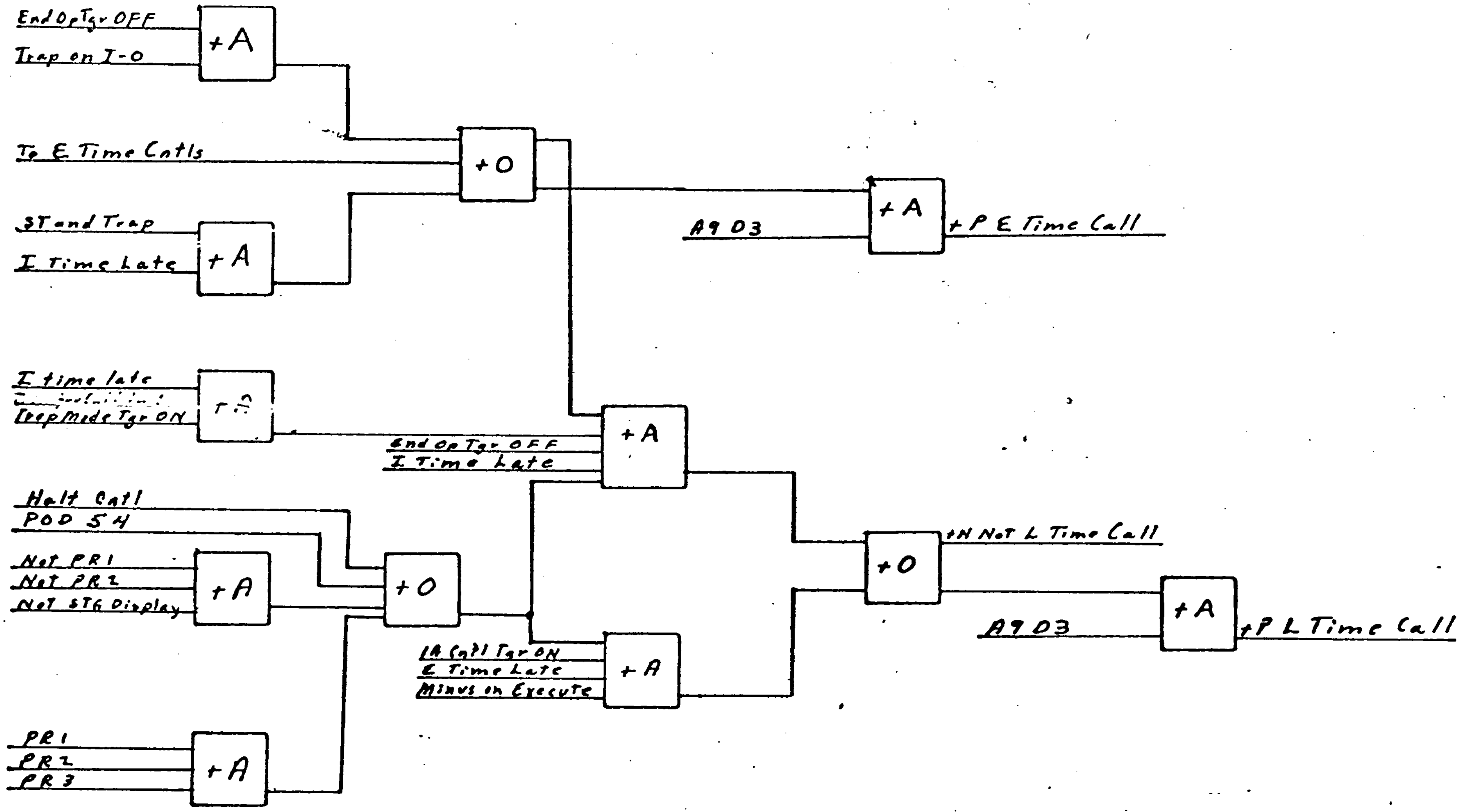


Figure 309. E and L time call

Go To I E and L time (Systems 08.00.12.1)

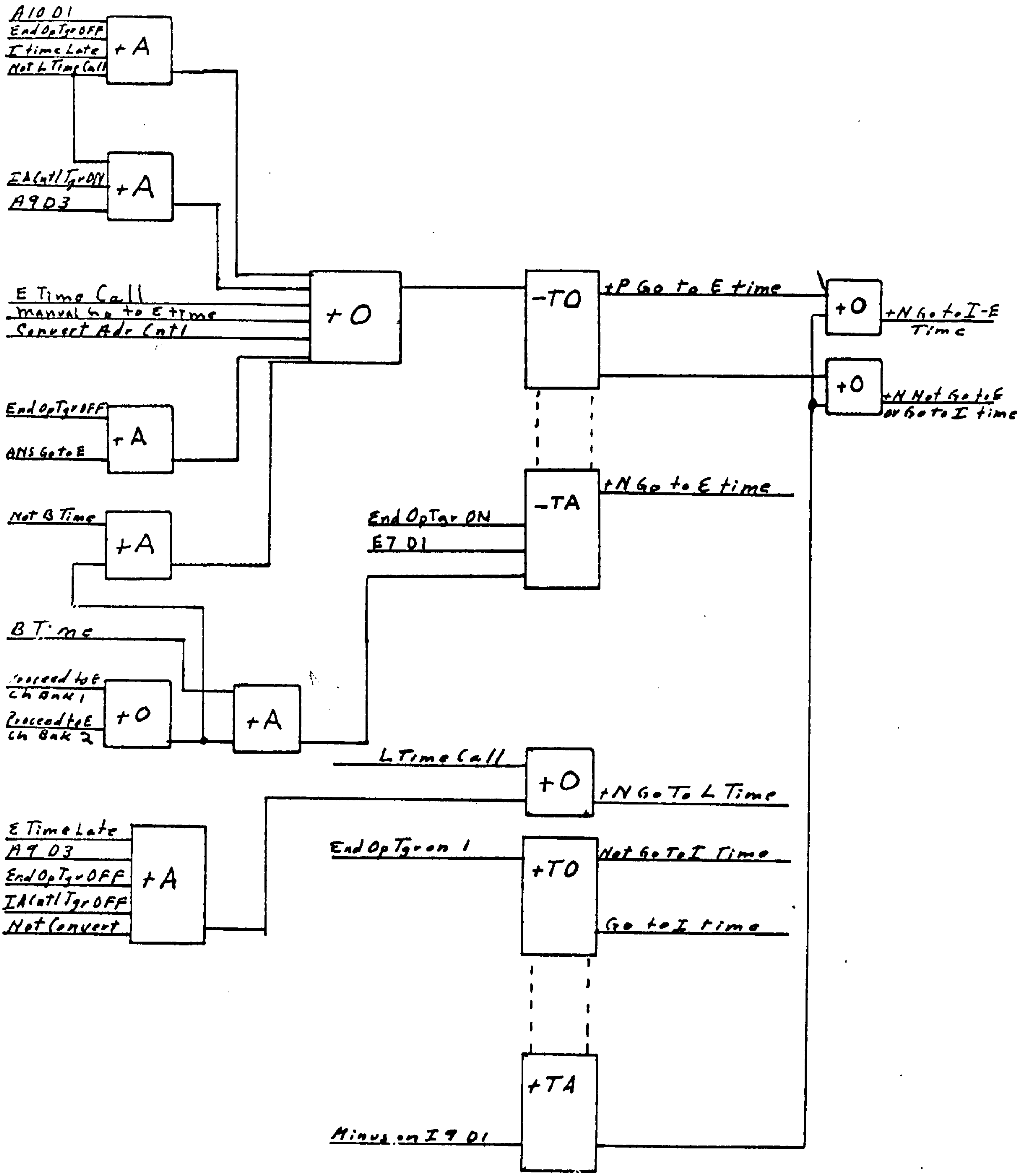


Figure 310 Go to I. E, L Time.

to I, E and L time" circuits to bring up control lines for the next ~~mix~~ cycle.

To summarize -- the following events occurred prior to and during the instruction cycle just described :

1. End operation trigger on at previous A10.

2. From A11 to I5 the new instruction is being located in storage.

3. Master I time trigger turned on at A11.

4. At I6 the new instruction is brought to the multiplexor storage bus input. Also at I6(D2), bitsS, 1-11 of the new instruction are gated to the program register for decoding.

5. At I7(D1) the incoming information is gated intothe storage register. Bits 21-35 continue through the storage register to the address register.

6. At I9(D2) the contents of the address register have been gated through the index adders for address modification and back to the address register.

7. At I9 and I, E or L time call is initiated, as determined by computer needs for the following cycle.

8. At I10 the end operation trigger is turned on (if this is a one cycle instruction).

9. At I11 the master I time trigger is turned off and the next machine cycle is started.

Instruction cycle timing is more complex than has just been described. The 7094 has a lookahead capability, enabling it to look at two instructions at once, one in the storage register and the other in the instruction backup register. The computer will alwyas bring two instructions from core storage, one at the even-numbered address and the other at the odd address. Under certain conditions these two instructions can be procwssæd under overlap conditions, reducing the number of machine cycles required. In general, this feature results in a savings of .4 of a cycle per instruction. Lookahead is presented later inlthis section.

Master E Time

The computer goes to E time when a reference to core storage is needed. For example, the instruction clear and add (CLA), requires two cycles of operation, an I and an E. At the completion of the I cycle the instruction has been decoded and the computer has the core storage address of the operand.

To complete the instruction it is necessary to locate and bring the operand out of core storage, to the storage bus, into the storage register, through the adders and finally, into the accumulator. When the operand is in the accumulators the instruction cycle for CLA is complete.

The first step in this sequence is to set the master E time trigger, Figure 311. This was begun at I9 by an E time call. Decoding the information in the [illegible] program register determined the need for an E cycle. The E time call signal, applied to the "go to I, E or L time" control circuits (Figure310) brings up the go to E time control signal line. This is one of the three inputs necessary, as shown in Figure 311, to set the Master E time trigger. The other two conditions are:

1. Not go g to I time.

Master E Time (Systems 08.00.19.1)
+ N

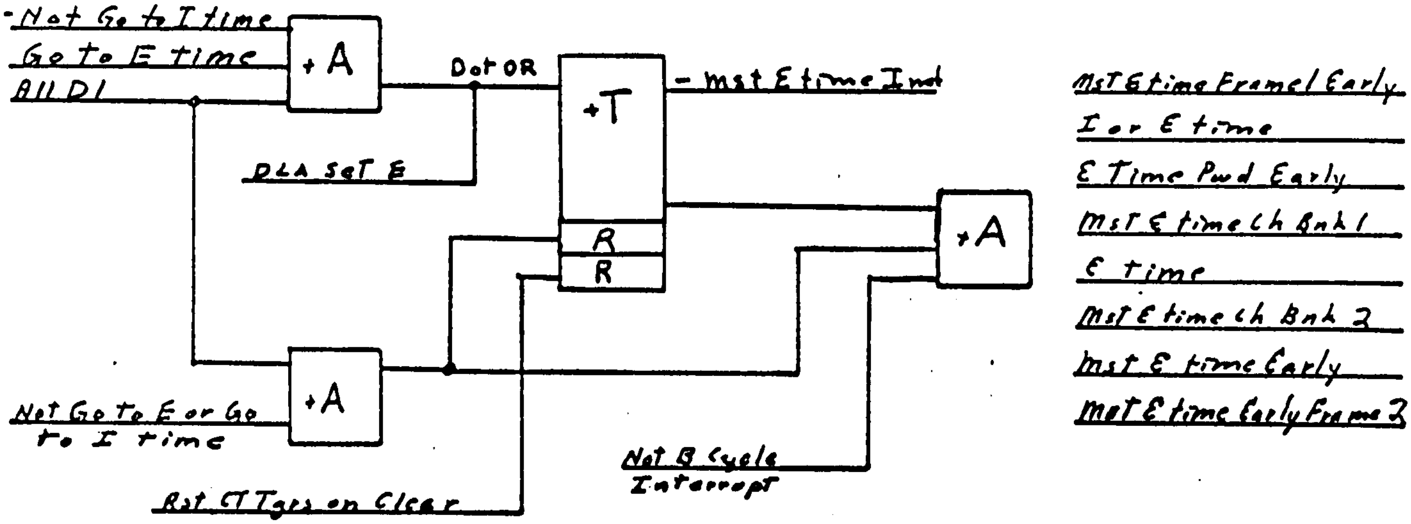


Figure 311 Master E Time

2. A11(D1)

"Not go to I time" is determined by instruction decoding and the three input conditions are met when the A11(D1) clock pulse rises. This sets the master E time trigger and the computer will operate in E time for at least one cycle.

As occurred during the I cycle, the computer uses the time from E11 to E6 to locate the CLA operand in core storage and bring it to the multiplexor storage bus input.

At E6(D2) the data word of the ~~operand~~ instruction (operand) is gated through the storage bus to the storage register. At the end of E8 the information is in the storage register.

At E9 action for the next cycle is initiated. This, again, may be an I, E or L cycle as determined by computer needs. For the CLA example, the next cycle would be an I cycle.

At E10 the end operation trigger is turned on (figure 307) and at E11 the computer will go to the next cycle. It is during this next cycle that the computer completes ~~the~~ the action begun by the E cycle. Information in the storage register is taken through the adders into the accumulator and the instruction CLA is completed.

Master L Time

L cycles are used by the computer during arithmetic and logic operations. For example, normalizing during a floating add instruction, i.e., shifting the AC left until position 9 contains a 1, is done during L time. Any computer function in which the contents of a register are manipulated, without a reference to core storage, is done in L time.

Figure 309 shows the conditions necessary to generate an L time

Master L Time (Systems 08.00.20.1)

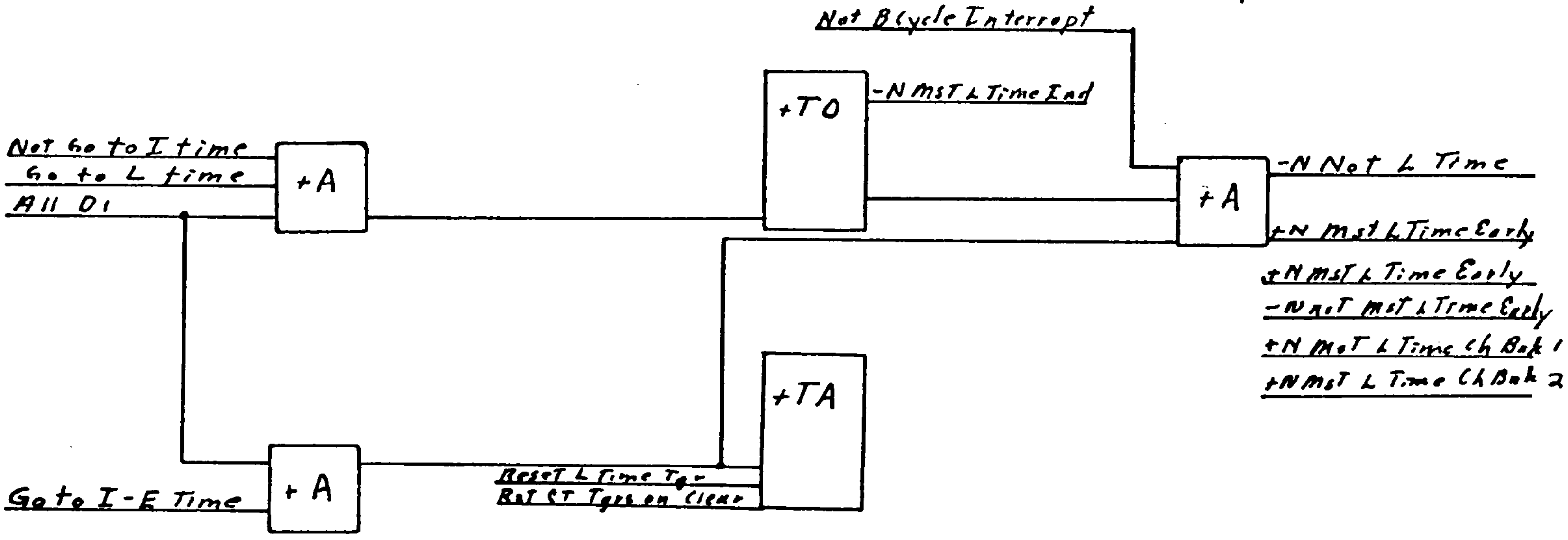


Figure 312. Master L time

call. Figure 310 shows how L time call results in a "go to L time " signal. This signal is one of the three requisites to turn on the master L time trigger, Figure 312. The other two requisites are not go to I time and A11(D1).

Once in L time, the computer will continue in L time until an internal request is generated to change to an~~d~~ I or E cycle. Generation of such a request is usually contingent upon the completion of an arithmetic or logic function.

For example, the shift counter is set, during a floating add instruction, to the number of shifts necessary to equalize the characteristics of the numbers being added. The shift counter is sensed after each shift (one for each clock pulse) to see if it is equal to 1 or 0. When either condition exists, and at L9, the end operation trigger is set and the instruction completed.

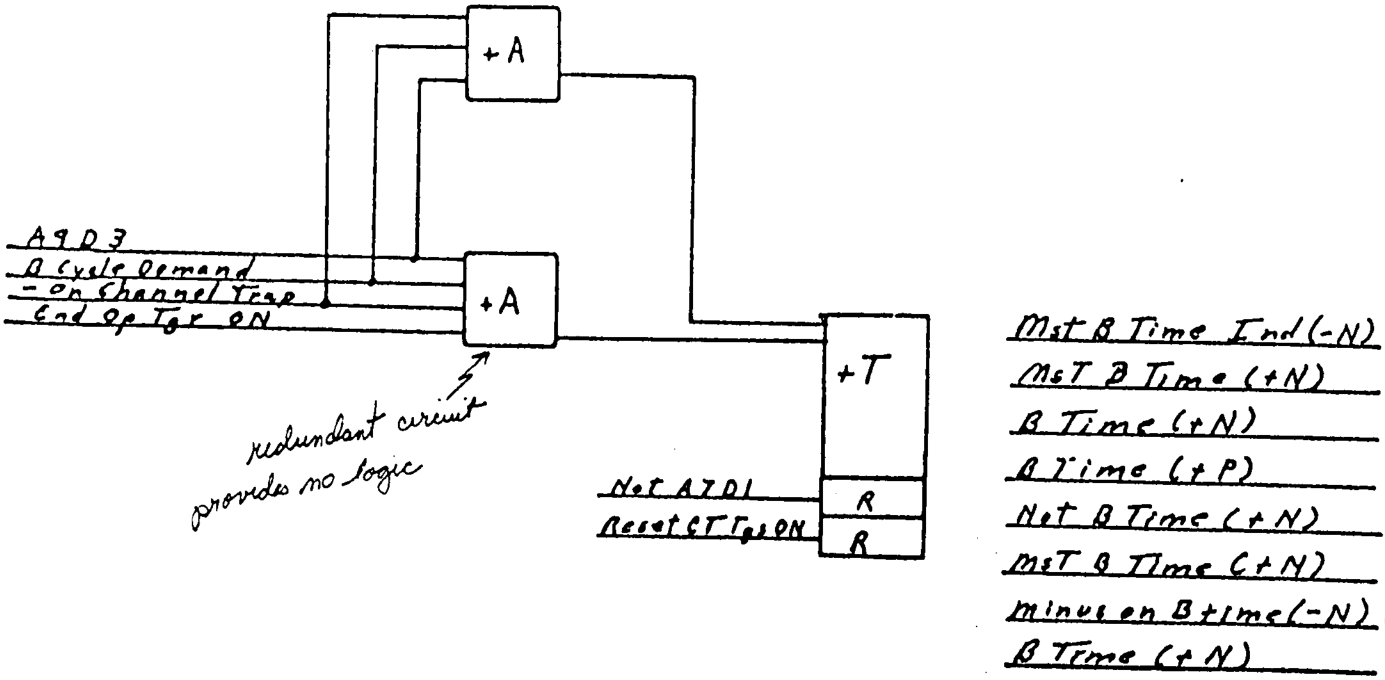
Other conditions for L time are given in this manual in the sections on arithmetic and logic operations.

Master B Time

During a B cycle a data channel is either taking information from, or putting information into a core storage location. Data channel demands must be recognized immediately by the computer because of the high speed I/O devices used with the 7094. Should a data channel encounter excessive delay, data might be lost.

A request for B time is recognized by granting a B cycle at the end of the machine cycle in which the request occurred. An exception to this occurs, as shown in Figure 313, when a B cycle der and is made at 9 time

Master B Time (Systems 08.00.21.1)
+N



redundant circuit provides no logic

NOTE : A B cycle demand and A9 time, when no channel trap exists, is a priority condition and will initiate B time without an end operation trigger.

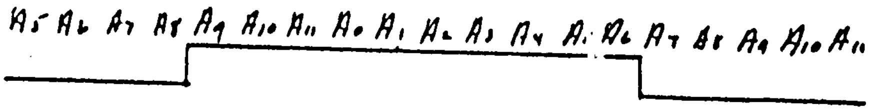


Figure 3/3
323 A

and the computer is not processing a channel trap. Note, also, that B time cannot be initiated when a channel trap exists.

If the cycle following the B cycle demand is to be an E or I cycle, computer operation will be interrupted until such time as no more B cycles are required. This is necessary because only one computer component can use core storage at any given time. Notice that even though the Master I time trigger is on (Figure 306) or the master E time trigger is on (Figure 311) a B cycle interrupt can block the outputs.

If the cycle following a B cycle demand is an L cycle, however, both B and L times occur simultaneously since the computer does not make reference to core storage during L cycles.

Fiigure 314 shows the conditions necessary to initiate~~d~~ a B cycle interrupt. This is the signal referred to above as contrdling the ~~mpx~~ outputs of the master I and E time triggers.

B Cycle Interrupt (Systems 08.00.13.1)

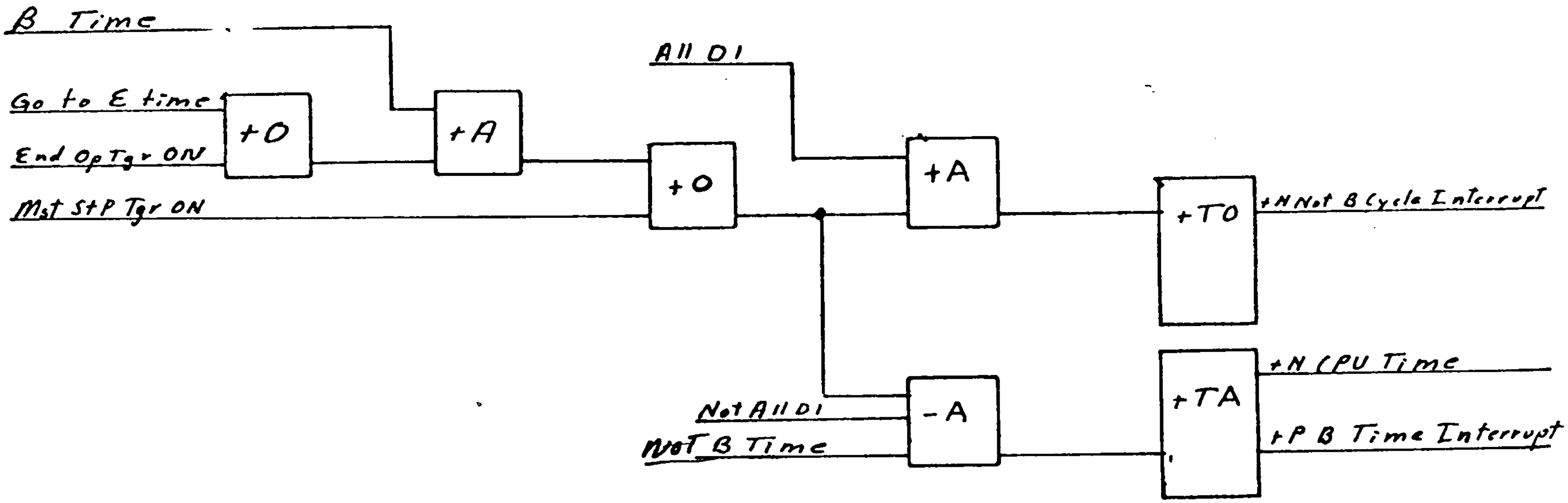


Figure 314

7094 Lookahead Capability

The 7094 always brings two instructions out of core storage, the even-numbered instruction called for by the program counter, and the next highest odd-numbered instruction. The even-numbered instruction is gated to the storage register and the odd-numbered instruction is held in the instruction back-up register (IBR).

While in the IBR the odd-numbered instruction is partially decoded to determine if itmeets certain criteria of overlap operation. If the logic circuits determine overlap is possible the computer performs a store lookahead (SLA), transfer lookahead (TLA) or data lookahead (DLA). If the instruction in the IBR cannot be overlapped, it is ignored and will be brought into the computer through the storage register on the next instruction cycle.

Because the IBR instruction is partially decoded during the even-numbered instruction cycle sequence the computer can act upon it earlier than usual and substantially reduce the number of cycles required for two instructions.

Suppose the instruction clear and add (CLA) were in storage location 100 and the instruction Add (ADD) were in location 101. Without lookahead the computer would bring in CLA, perform the instruction, and then bring in ADD and perform the instruction. This would require a total of four cycles, and I and E cycle for each instruction.

```
/                I    /          E          /          I          /      E      /
/--------------CLA--------------------/------------------ADD-----------/
```

With lookahead, the full 72 bits of the two instructions are read out of core storage, CLA going to the storage register and ADD to the IBR.

the ADD instruction is decoded and, because the instruction calls for data from core storage the computer will enter the data lookahead mode. As soon as the CLA instruction is complete the computer will switch to ADD and bring the ADD operand out of storage. This overlapping reduces the number of cycles required for the two operations from four to three.

```
/          I          /          E          /     I          E          /     I
----------------CLA--------------------
                         ----------------ADD-------6/7-----------
```

Notice that the total elapsed time for the two instructions is three cycles. In the middle of the second I time, at I6, the computer was forced to go to E time. This was made possible by decoding the ADD instruction while it was in the IBR, a sequence normally requiring a full I cycle(this includes the time necessary to bring the instruction out of storage).

Timing rules for these instruction overlaps are as follows:

1. Data lookahead -- if the next instruction in the even location has a cycle [redacted] time of two or more cycles(excluding 06xx store type codes) and is indexable; and the instruction in the next higher (odd) location requires a data fetch (not a store operation) from core storage, the execution time of this next instruction is reduced by one cycle.

2. Store lookahead -- if the instruction in the even location has an octal code of 06xx (store type), the execution time of the next instruction is reduced by one cycle. If the instruction following an 06xx octal code instruction is a one cycle instruction it is [redacted] executed in overlapped operation and requires no cycle time.

3. Transfer lookahead -- if the instruction in the even location has an octal code of 02xx, 03xx, 04xx or 05xx (except 052x and 053x) and the next instruction is TRA, TTR, TXI, TIX, TNX, TXH, TXL or TSX; this instruction is executed in overlapped operation and requires no cycle time.

There are some exceptions to the above rules because of logical and compatibility reasons. For example, if the address of an 06xx instruction located at address 2n is 2n+1, no overlap is possible. If a multiple tagged index modification instruction is in an odd location, no overlap occurs. Overlap is not effective for rules 1 and 3 if the instruction in the even location is DVH, XEC, FDH, VDH, HPR, LCH, is non-indexable, or is a double precision floating point instruction. Research on instruction mixes indicates that out of 100 instructions executed, between 40 and 45 instruction fetches are eliminated if the above rules are ~~so~~ not applied. A rule of thumb is to reduce cycle time for all instructions executed by 0.4 cycles (0.8 microseconds).

A more complete discussion of lookahead, complete with flow and sequence charts, is given in this manual in the section on Instruction~~s and Data Flow.~~ Overlap.

401 7094 ARITHMETIC

401

BINARY ARITHMETIC

Binary arithmetic includes addition, subtraction, multiplication and division. Each is discussed in detail in the following paragraphs.

401.1 ~~2.1.1~~ ADDITION

Binary addition is simple. Its rules are as follows:

```
0 + 0 = 0
1 + 0 = 1
0 + 1 = 1
1 + 1 = 0 + 1 to carry
```

These rules operate in all cases of addition and apply to both addition of integers and of fractions. Binary numbers are added from right to left, and the carry is added to the adjacent bit on the left. The following examples illustrate the rules for binary addition. Note that the carry is placed in the column, to which it will be added, in parentheses.

```
         (1)          (11)    (1)     (1)

  0       1     10     11     100     101
+ 1     + 1   +  1   +  1   +   1   +   1
---     ---   ----   ----   -----   -----
  1      10     11    100     101     110
```

The technical terms in addition are defined as the augend, addend, and the sum. The augend is the term that is to be increased; the addend is the term, to be added to the augend; the sum is the result of the operation. For example:

```
  101   Augend
+ 011   Addend
-----
 1000   Sum
```

In adding more than one number, the addition of the first set of numbers is performed and, to the sum, is added the third number. To the sum of the succeeding additions, add the next number until all the numbers have been totaled. For example, add:

```
a.  011
    111
  + 110
  -----

    Addition of the first          011
    set of numbers               + 111
                                 -----
    First Sum                     1010
    Addition of the third
    number                       + 110
                                 -----
    Final Sum                    10000
```

```
b.  1101
    1001
    0010
  + 1111
  ------

    Addition of the first         1101
    set of numbers               +1001
                                 ------
    First Sum                    10110
    Addition of the third
    number                       +0010
                                 ------
    Second Sum                   11000
    Addition of the fourth
    number                       +1111
                                 ------
    Final Sum                   100111
```

Binary fractions are added in accordance with the rule that governs whole numbers. The binary point is fixed as in the decimal system. The carry from the addition of the binary fractions in the first position to the right of the binary point is an integer. For example, in addition of the following fractions:

```
a.  Decimal                    Binary
       1/8                       .001
       3/8                       .011
    -------                    ------
       4/8 or .5                 .100

b.     4/8                       .10
       6/8                       .11
    -------                    ------
     10/8 or 1.25               1.01

c.    5 3/8                    101.011
      6 7/8                    110.111
    -------                    -------
    12 2/8 or 12.25           1100.010
```

401.2 ~~2.1.2~~ SUBTRACTION

The rules for binary subtraction are as follows:

0 - 0 = 0
0 - 1 = 1 (borrow 1 and make 0 = 10) Read as one zero
1 - 0 = 1
1 - 1 = 0

The technical definitions of the terms used in subtraction are minuend, subtrahend, and difference. The minuend is the number to be decreased; the subtrahend is the quantity of the decrease; the difference is the result of the operation. Thus:

$$\begin{array}{rl} 0110 & \text{Minuend} \\ \underline{100} & \text{Subtrahend} \\ 010 & \text{Difference} \end{array}$$

The similarity which exists between decimal and binary arithmetic when a carry is involved is analogous to the similarity which exists when a borrow is involved. When subtracting a 1 from a 0, a 1 must be borrowed from the next higher order, diminishing that order by 1.

The following examples illustrate the rules for binary subtraction and the method of borrowing from the next higher order.

$$\begin{array}{r} 1101 \\ -\underline{0100} \\ 1001 \end{array} \qquad \begin{array}{r} 1110 \\ -\underline{0101} \\ 1001 \end{array} \qquad \begin{array}{r} 1100 \\ -\underline{1001} \\ 0011 \end{array}$$

In the first example, the subtraction of 0 from 1, 0 from 0, and 1 from 1 produces the difference. In the second example, a 1 must be borrowed from the second order when attempting to subtract the 1 of the first order from 0. The 1 in the second order then diminishes to 0. In the third example, a slightly different borrow situation arises. The 1 to be borrowed must come from the third order of the minuend. That 1 then diminishes to 0. The 1 of the first order of the minuend can then be borrowed from the 10 which appears in the second order. Borrowing the 1 from 10 leaves a 1 in the second order of the minuend. Applying the rules of binary subtraction then produces the difference shown.

401.3.1 ~~2.1.2.1~~ Complement Method

The preceding discussion delineated the methods of direct subtraction. The complement method of subtraction is a means of subtracting by addition. Design requirements of a processing unit do not allow for borrowing, so the complement method of subtraction fits in with processing unit design and capabilities.

A disadvantage of direct binary subtraction is that the direct subtraction of a number from a smaller number yields an incorrect result unless the subtraction is done by subtracting the smaller from the larger and then changing the sign of the difference. For example:

$$\begin{array}{r} 5/16 \\ -\,9/16 \\ \hline -\,4/16 \end{array} \qquad\qquad \begin{array}{r} 0.0101 \\ -\,0.1001 \\ \hline ? \end{array}$$

The difficulty encountered with negative results and the problem of providing for borrowing in circuit design are eliminated by correcting the subtraction to an addition of negative numbers by means of the complement process.

The complement system of subtraction is possible because it is possible to limit the number of significant digits to be used in any one problem or machine. The problem is then said to have a modulus, which is the count of the maximum number of numbers it would be possible to represent in this problem. For instance, suppose that a binary machine has facilities for handling 4 places. The machine could represent 16 different numbers from 0 to 1111. Such a machine has a modulus of 16 and is said to perform modulo 16 arithmetic.

The significance of the modulus of the machine is that each time an addition results in a number equal to ~~a~~ or greater than the modulus of the machine, an integral multiple of the modulus is lost. An example of this action in everyday life is given by the automobile speedometer. When it reaches 100,000 miles, it resets to zero and starts over. The speedometer has lost 100,000 by resetting to zero. This property of machine-counting methods is important in the use of complements for subtraction by addition.

The complement method of subtract may be derived from the following identity:

$$P - M + (M - N) = P - N$$

P = Minuend
N = Subtrahend
M = Modulus of the machine
$P - N$ = Difference sought

To derive the complement system of subtraction, let $(M - N)$ equal a number called the complement of N. Let C stand for this complement so $M - N = C$. Now substitute C in the identity:

$$P - M + C = P - N$$

or $$(P + C) - M = P - N$$

If M is moved to the other side of the identity, it becomes:

$$P + C = M + (P - N)$$

It is now evident that the minuend plus the complement of the subtrahend is equal to the difference of the minuend and subtrahend plus the modulus. It should now be recalled that when two numbers are added to obtain a sum greater than the modulus, the modulus is lost. Therefore, $P + C = P - N$ in any system with a fixed modulus, provided only that the sum $P + C$ is greater than the modulus of the number system used.

The above is a derivation of what, in binary arithmetic, is called the 2's complement system. A similar derivation of a 1's complement system may be derived using $(M - 1)$ in place of M. In this case, the final equation is $P + C_1 - 1 = P - N$, which implies that the difference sought will be found by adding 1 to P + C_1. Note that C_1 is equal, in this case, to (M - 1) - N.

401.2.2
~~3.1.3.8~~ 1's Complement

Every processing unit has a modulus which is one greater than the largest number the processing unit can register. For example, a 6 place binary counter can express all the numbers from 0 to 111111. The modulus of such a counter is 1 000 000.

To obtain the 1's complement of a number, it was shown in the derivation above that the number must be subtracted from (M - 1). Therefore, to obtain the 1's complement of a number in a 6-place machine, the number is subtracted from (1 000 000 - 1); that is, from 111 111. As an example, find the 1's complement of the binary numbers 101 001 and 001 101:

111 111	Modulus - 1
101 001	Number
010 110	1's complement of number

111 111	Modulus - 1
001 101	Number
110 010	1's complement of number

A close examination of the numbers and their 1's complements shows that the 1's complement in binary arithmetic is nothing more than the original number with its bits reversed. That is, the original number's 0's are made 1's and the original number's 1's are made 0's. The way to get the 1's complement, then, is by inspection; just exchange 0's for 1's and 1's for 0's. For example:

100 101 = Number
011 010 = 1's complement

To perform subtraction by the 1's complement method, proceed as follows:

1. Find the complement of the subtrahend.

2. Add the complement to the minuend.

3. Perform "end-around carry" if there is a carry out of the highest position of the difference (explained below).

The result is the difference in complement form if it is negative and in true form if it is positive.

There are four possibilities, as shown by the examples below. All except the last are treated exactly the same. The last requires the extra step of end-around carry, which is a carry from the highest order around to the lowest order. This carry is required because of the cyclical nature of the number system. The only time it is required is when the minuend is larger than the subtrahend, that is, when the answer will come out a true positive answer. Fortunately, whenever it is required, there is a carry from the left-most position, which serves as a reminder.

Examples	Direct Subtract		Complement Subtract	
Minuend < Subtrahend	+ 011 011	Minuend	011 011	Minuend
	- 101 010	Subtrahend	010 101	Complement
	- 001 111	Difference	110 000	Complement of Difference
Minuend = Subtrahend	+ 011 011	Minuend	011 011	Minuend
	- 011 011	Subtrahend	100 100	Complement
	000 000	Difference	111 111	Complement of Difference
- Minuend > - Subtrahend	- 011 011	Minuend	100 100	Minuend Complement
	- 010 011	Subtrahend	010 011	Subtrahend
	- 001 000	Difference	110 111	Complement of Difference
Minuend > Subtrahend	011 011	Minuend	011 011	Minuend
	- 010 101	Subtrahend	101 010	Complement
	+ 000 110	Difference	000 101	Difference - 1
	with a 1 end carry	------>	1	
			000 110	True Difference

401.2.3 ~~2.1.2.3~~ 2's Complement

In the dorivation of the complement system, it was shown that a 2's complement of a number is equal to the modulus minus the number, (M - N). Therefore, to obtain a 2's complement in a 6-place machine, the number is subtracted from the modulus, 1 000 000. As an example, find the 2's complement of the numbers 101 001 and 001 101:

```
1 000 000 = Modulus              1 000 000 = Modulus
  101 001 = Number                 001 101 = Number
  010 111 = 2's Complement         110 011 = 2's Complement
```

An examination of the numbers and their complements shows that the 2's complement of a number is the same as the 1's complement with a 1 added to it. The 2's complement is therefore formed by obtaining the 1's complement and adding 1 to it. For example, to form the 2's complement of 001 101:

```
001 101 = Number
110 010 = 1's Complement
110 010 = 1's Complement
    + 1
110 011 = 2's Complement
```

To perform subtraction by the 2's complement method:

1. Find the 2's complement of the subtrahend.
2. Add this complement to the minuend.

The result is the difference in complement form if it is negative and in true form if it is positive. In the 2's complement system, there is no need to end-around carry.

401.2.4 ~~2.1.2.4~~ Signed Numbers

How can negative numbers in complement form be distinguished from positive numbers in true form. In this regard, also, binary numbers offer an advantage with respect to representation. The sign of a number is binary in nature; that is, a number is either positive or negative. Thus, a bit representing the sign can be used in addition to the bits representing magnitude. A 0 in the sign bit position can be interpreted to mean that the number is positive. A 1 in the sign bit position can be interpreted to mean that the number is negative. By treating the signs separately from the magnitudes in each operation, the result sign can be predicted. Therefore, the rules of algebra apply in determining the result sign.

401.3 ~~2.1.3~~ MULTIPLICATION

The rules for binary multiplication are similar to those of decimal multiplication. The rules for multiplying two single digits are the same in both systems. These rules are:

0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1

The general procedure when multiplying two multiple digit binary numbers is the same as that in decimal arithmetic. That is, the multiplicand is multiplied by a digit of the multiplier, and the partial product obtained is placed so that the least significant digit is under the multiplier digit. When all the partial products have been found, they are added together to find the final product. The only difference between decimal and binary multiplication, therefore, is in the summing of the partial products. In binary, the binary addition table is used while in decimal, the decimal table is used.

As can be seen from the following examples, the method of obtaining partial products and then adding them to obtain the final product is identical to that of decimal arithmetic.

Multiplicand	1010	10.11	1111
Multiplier	1101	100.1	1111
First Partial Product	1010	1 011	1111
Second Partial Product	0000	00 00	1111
Third Partial Product	1010	000 0	1111
Fourth Partial Product	1010	1011	1111
Final Product	10000010	1100.011	11100001

Note the placement of the binary point in the second example. The same rules hold for its placement as hold for placement of the decimal point in decimal arithmetic.

The third example also illustrates an interesting point. This is the multiplication of the two largest possible 4-bit numbers. The product is 8 bits long. In other words the largest product that can result from the multiplication of two numbers will be no longer than the sum of the number of bits in the multiplier and multiplicand.

If a number is multiplied by the radix of the number system, this multiplication has the effect of shifting the number one place to the left with respect to the radix point. This is true in any number system. For example, multiply 12.51_{10} by 10 (the radix of the decimal system) and multiply the number 10.11_2 by 2 (the radix of the binary system):

Number	12.51	10.11
Number Times Radix	125.1	101.1

Binary multiplication then is nothing more than a series of add and shift operations. An example of such an operation is given under Fixed Point Arithmetic..

~~2.1.4~~ 401.4 DIVISION

Binary division is the process of counting the number of times a divisor goes into a dividend. The count of the number of times the divisor may be subtracted from the dividend before a negative remainder occurs is called the quotient.

Direct binary divisi[illegible] is performed by a series of subtractions of the divisor (actually a multiple of the divisor), just as it is in the decimal system. For example, divide 100 011 100 by 1110:

```
                  bd chi jk
                  10 100.01
               -----------------
1110          | 100 011 100.00
(a)              11 10
               ------------
                (c)  111 1
                (f)  111 0
                    ------------
                (g)       100 00
                (l)        11 10
                         -------
                (m)           10
```

In tho example, the first step is to place the divisor below the dividend in a position which is as far removed to the left as possiblo (a), but which will allow a position differenco to result when the divisor is subtracted from tho dividend. Since tho divisor will go into this many bits of the dividond onco, a 1 is placed in tho quotient at b in the samo column as the lowest order digit of the divisor. Tho divisor is thon multiplied by tho quotient digit, and the resulting product is subtracted from the dividend to produce the positive difference (c) called the current remainder. The next digit in the dividend is brought down to line c. Compare the divisor to line c; note that the divisor is larger than line c, or that the divisor goes into line c 0 times. Thorefore, placo a 0 in the quotient at the d position. The next digit of the dividend is then brought down to line c. Comparing the divisor to line c shows line c to be greater. Place a 1 in tho quotient at tho e position. Multiply the divisor by the last quotient bit to form line f. Subtract line f from lino c to start line g. The next digit in the dividend is brought down to lino g. Compare tho divisor to line g; the divisor is greater, so place a 0 in the quotient at powition h. Bring the next digit of the dividend down to line g; by comparison line g is still smaller than the divisor. Place a 0 in the quotient in position i, and place the next dividend digit on line g. Still, line g is smaller than the divisor, so a 0 is placed in the quotient at position j. Placing the next dividend digit on line g now makes line g greater than the divisor. Place a 1 in the quotient at position k, and multiply the divisor by this 1 to form line l. Subtract line l from line k to start line m. Assuming a quotient has been developed of sufficient length, terminate the operation. The quotient is 10100.01 with a remainder of 10 (line m).

Since the quotients bit is always either 0 or 1, the division process can be reduced to a series of subtractions of the divisor, multiplied by the power of the quotient bit being sought from the dividend. Each time a subtraction resulted in a positive current remainder, a 1 would be placed in the corresponding quotient bit po~~w~~sition, and the process is immediately repeated for the next quotient bit. Each time the subtraction results in a negative remainder, a 0 is placed in the corresponding quotient bit. In this case, the current remainder is restored to a positive number by adding the divisor back to it. Following this, the next quotient bit is obtained by the subtraction of the divisor multiplied by the power of the next quotient bit.

Since the quotient bits are generated from left to right, the power of each quotient bit is one smaller than that of the last bit generated. This means that as the divisor is successively subtracted from the dividend (or current remainder), the divisor is shifted to the right in relation to the binary point. The division process can therefore be reduced to a process of successive subtract and shift steps. An example of such a process is given under Fixed Point Arithmetic.

FIXED POINT ARITHMETIC

Fixed point arithmetic is the most basic form of arithmetic. Simply stated, it is the process of computation using quantities whose magnitude is completely expressed by a single value field. The relationship of the magnitude to zero is expressed by a sign position. In fixed point arithmetic, the length of an operand is generally determined by the smallest unit of data that can be accessed in core storage. In the 7094, fixed point arithmetic operands have the following basic format:

S	VALUE FIELD
	1 35

The sign bit S determines whether the magnitude is positive or negative. When S is a 0, the magnitude is positive; when S is a 1, the magnitude is negative. The value field is 35 bits long and states the magnitude of the number. A fixed point operand can then be defined as a unit of data 36 bits long, containing a sign bit and 35 magnitude bits.

Fixed point arithmetic in 7094 includes addition, subtraction, multiplication and division. All operations involve only two operands. One operand is explicitly addressed, and one operand is implied. In addition and subtraction, the explicitly addressed operand is obtained from the core storage location specified by the instruction word effective address. The implied operand is obtained from the accumulator. The former is generally known as the addressed operand; the latter, as either the implied or accumulator operand. In the computer, the addressed operand is placed into the Storage Register, which has the format shown above. The implied or accumulator operand has the following format:

VALUE FIELD

S	Q	P	1 89P9 35

The accumulator value field is 38 bits long. The additional bits, Q and P and 9P are provided primarily to handle conditions which result in a carry of 1 out of position 1 and position 9. Bits P and Q are therefore known as overflow bits and are treated as the two highest order accumulator bits during the execution of fixed point arithmetic. Position 9P is not used in fixed point arithmetic.

The actual arithmetic takes place in the adder which has the following format:

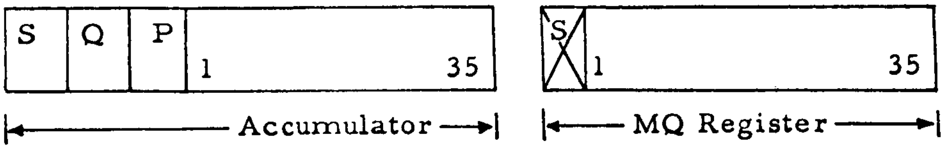
Q	P	1 89Q9P9 35

Note: 9Q and 9P not used in fixed point

Basically, the contents of the storage register are transferred into the adders simultaneously with the transfer of the accumulator contents to the adders. An addition or subtraction is effected, and the result is transferred into the accumulator.

In multiplication, the addressed operand is obtained from the core storage location specified by the instruction word effective address; the implied operand is obtained from the multiplier - quotient (MQ) register. In the computer, the addressed operand is placed in the storage register, which has the basic format of a sign bit and a 35-bit value field. Storage register contents become the multiplicand. MQ register contents form the multiplier, which has a format identical with the multiplicand. Multiplication is effected by a combination of right shifts and simple additions. A multiplication result is placed in the combined accumulator - MQ register, with MQ register bit 35 the lowest order bit. Multiplication is algebraic, and the result sign is placed in both the accumulator sign position and the MQ register sign position.

In division, the addressed operand is obtained from the core storage location specified by the instruction word effective address; the implied operand is obtained from the combined accumulator - MQ register. The addressed operand is placed in the storage register and becomes the divisor; the combined accumulator - MQ register becomes the dividend. Divisor format is the basic single sign bit and 35 value field bits. The dividend format is a single sign bit and 72 field bits:



The result or quotient is placed in the MQ register and has a format identical with the divisor. Remainder bits, if any, go into the accumulator, with a format of one sign bit and 37 value field bits; accumulator bit 35 is the lowest order remainder bit. Division is effected by a combination of subtractions and left shifts.

Addition

In performing addition in the 7094, the general rules of algebra must first be applied to the signs of the quantities involved to determine whether the sum or difference of the quantities involved is to be obtained. Therefore, when adding two positive quantities, the result is the sum of those quantities with a positive sign. When adding a positive and a negative quantity, the sum is actually the difference of the two quantities with the result sign being the sign of the larger magnitude. Finally, when adding two negative quantities, the result is the sum of the quantities with a negative sign.

Assume the quantity $+200_8$ is to be added to the accumulator, which contains $+75_8$. The result is $+275_8$. To satisfy machine operand format, convert the quantities to their binary equivalents:

a. $+200_8 = +010\ 000\ 000$

b. $+\ 75_8 = +000\ 111\ 101$

Insert these binary numbers into respective data words with the lowest order bit going into bit 35:

	0	0 ←——→ 0	0	1	0	0	0	0	0	0	0
a.	S	1 26	27	28	29	30	31	32	33	34	35

Storage Register

	0	0	0	0 ←——→ 0	0	0	0	1	1	1	1	0	1
b.	S	Q	P	1 26	27	28	29	30	31	32	33	34	35

Accumulator

Bits 1 through 26 are not needed to express the quantities and are therefore all zeros. Because accumulator bits Q and P are treated as part of the value field and the accumulator value is assumed as $+75_8$, bits P and Q are zeros. Since each number is positive, a 0 is placed in the respective sign bit S.

Fixed point addition in the 7094 is identical with that described in binary addition: 0 + 0 = 0; 0 + 1 = 1; 1 + 1 = 0 with a 1 carry to the next higher position. Adding the two operands produces a result magnitude of

010 111 101, with a result sign of 0. In machine operand format the result is illustrated as follows:

0	0	0	0 ←——→ 0	0	1	0	1	1	1	1	0	1
S	Q	P	1 26	27	28	29	30	31	32	33	34	35

Accumulator

If the same magnitudes are used but the signs changed to negative, the entire handling of the magnitude remains unchanged in performing the addition. The 7094 treats the sign bits separately. To correctly represent the negative values, simply insert a 1 in the sign bit position of each of the operands and the result; this is what is done in the computer.

Since algebraic principles are employed, addition of two quantities with unlike signs is effectively a subtraction. Using the same values, but changing the sign of the accumulator operand to a minus, the problem becomes $(+200_8) + (-75_8)$. To accomplish addition, line up the octal points and subtract:

$$\begin{array}{rr} & +\ 200_8 \\ + & \\ & -\ 075_8 \\ \hline & +\ 103_8 \end{array}$$

To satisfy machine operand format, convert the values to their binary equivalent:

a. $+200_8 = +010\ \ 000\ \ 000$

b. $-075_8 = -000\ \ 111\ \ 101$

Insert these binary numbers into their respective data word with the lowest order bit in each value going into bit 35:

a.

0	0 ←	→ 0	0	1	0	0	0	0	0	0	0
S	1	26	27	28	29	30	31	32	33	34	35

Storage Register

b.

1	0	0	0 ←	→ 0	0	0	0	1	1	1	1	0	1
S	Q	P	1	26	27	28	29	30	31	32	33	34	35

Accumulator

Bits 1 through 26 are not needed to express the quantities and are therefore all zeros. Accumulator bits Q and P are implied zeros by the assumed accumulator value.

The machine effects the addition of values having unlike signs as follows:

1. Complement the accumulator value field.
2. Add the complemented accumulator value field and storage register value field.
3. Place the result in the accumulator.
4. Compare the accumulator and storage register signs:
   a. If alike, check for a carry out of value field position 1. The coincidence of like signs and a carry out of position 1 indicates an overflow.
   b. If unlike, check for a Q carry:
      1) If there is a Q carry, add 1 to the accumulator in the lowest order position (bit 35), invert the accumulator sign, and place the resultant operand in the accumulator.
      2) If there is no Q carry, complement the accumulator value field.

Following the above procedure, the addition is performed as follows:

1. Storage Register = $+200_8$ = +010 000 000

   Accumulator = -075_8 = -000 111 101

2. Complementing the accumulator value field results in its containing 111 000 010, with bits Q - 26 all ones.

3. Add:

   ```
   010 000 000
   111 000 010
   -----------
   001 000 010
   ```

   with a 1 carry propagated throughout the rest of the bits (Q - 26) and out of Q.

4. Placing the intermediate result into the accumulator, it now contains -001 000 010. Bits Q - 26 are all zeros because of the propagated carry.

5. Checking the accumulator and storage register signs reveals they are unlike.

6. Checking for a Q carry reveals one.

7. Adding 1 to the accumulator lowest order bit makes the value field 001 000 011, and inverting the sign makes it positive (0).

8. The resultant value in the accumulator is +001 000 011, which equals $+103_8$.

Repeating the problem with $+200_8$ the accumulator operand and -75_8 the addressed operand causes the following:

1. Storage Register = -75_8 = -000 111 101

   Accumulator = $+200_8$ = +010 000 000

2. Complementing the accumulator value field results in its containing 101 111 111, with bits Q - 26 all ones.

3. Add:

```
        000 111 101
        101 111 111
        -----------
        110 111 100  with bits Q - 26 unaffected.
```

4. Placing the intermediate result into the accumulator, it now contains +110 111 100. Bits Q - 26 are all ones.

5. Checking the accumulator and storage register signs reveals they are unlike.

6. Checking for a Q carry reveals none.

7. Complementing the accumulator value field yields a final result of +001 000 011.

The term overflow means that the capacity of the machine has been exceeded. The arithmetic result cannot be represented by the machine, because it contains more than 35 value field positions. It was previously stated that accumulator bits Q and P are called overflow bits. The name, however, only provides an easy means of identifying these bits as a pair. Because they could originally contain 00, 01, 10, or 11, their significance depends on the problem. When dealing with values having like signs, a resultant 1 in either bit or in both bits indicates an overflow. In this case, the overflow is recorded, but subsequent action depends on the program being executed. When dealing with unlike signs, the overflow bits are significant as a pair, and in this sense, they either generate a Q carry or they don't generate a Q carry. If a carry is generated, it indicates (1) that the accumulator operand was the smaller operand and (2) that the number presently in the accumulator value field is a true number equal to one less than the correct answer. If a Q carry is not generated its absence indicates (1) that the accumulator operand was the larger operand and (2) that the number presently in the accumulator value field is the correct answer in complement form.

Subtraction

Subtraction in the 7094 is algebraic and is accomplished by complement addition. The procedure is as follows:

1. Complement the storage register sign.
2. Compare the accumulator and storage register signs:
   a. If alike, add the accumulator and storage register.
   b. If unlike, complement the accumulator, and then add the complemented accumulator and storage register.
3. Place the addition result in the accumulator.
4. Compare the accumulator and storage register signs:
   a. If alike, check for a carry out of value field position 1. The coincidence of like signs and a 1 carry out of value field position 1 indicates an overflow.
   b. If unlike, check for a Q carry:
      1) If there is a Q carry, add 1 to the present accumulator value field in the low order position, and invert the accumulator sign.
      2) If there is no Q carry, complement the accumulator value field.

Assume the problem $(+600_8) - (-55_8)$ where $+600_8$ is the addressed operand and -55_8 is the implied operand. The result is -655_8. To satisfy machine operand format, convert the quantities to their binary equivalents:

a. $+ 600_8 = + 110\ 000\ 000$

b. $- 55_8 = - 000\ 101\ 101$

Insert these binary numbers into respective data words with the lowest order bit goint into bit 35.

a.

0	0	←→	0	1	1	0	0	0	0	0	0	0
S	1		26	27	28	29	30	31	32	33	34	35

REGISTER

b.

1	0	0	0	←→	0	0	0	0	1	0	1	1	0	1
S	Q	P	1		26	27	28	29	30	31	32	33	34	35

ACCUMULATOR

Bits 1 through 26 are not needed to express the quantities and are therefore all zeros. Because accumulator bits Q and P are treated as part of the value field and the accumulator value is assumed as $- 55_8$, bits P and Q are zeros. The addressed operand is positive, so its sign bit is a 0, whereas the implied operand is negative, so its sign bit is a 1.

Following the procedure, the subtraction is accomplished as follows:

1. Storage Register $= + 600_8 = + 110\ 000\ 000$

   Accumulator $= - 55_8 = - 000\ 101\ 101$

2. Complementing the storage register sign results in its containing $- 110\ 000\ 000$.

3. Comparing the operand signs reveals they are alike.

4. Add:

$$\begin{array}{r} 110\ 000\ 000 \\ \underline{000\ 101\ 101} \\ 110\ 101\ 101 \end{array}$$

   with bits Q - 26 all zeros.

5. Placing the addition result in the accumulator, it now contains $- 110\ 101\ 101$

6. Comparing the accumulator and storage register signs reveals they are alike.

7. Checking for a Q carry reveals none.

8. The final answer in the accumulator is - 110 101 101 which equals - 655_8.

Repeating the problem, but with - 55_8 the addressed operand, the operand formats are as follows:

1	0 ← → 0	0	0	0	1	0	1	1	0	1
S	1 26	27	28	29	30	31	32	33	34	35

STORAGE REGISTER

0	0	0	0 ← → 0	1	1	0	0	0	0	0	0	0
S	Q	P	1 26	27	28	29	30	31	32	33	34	35

ACCUMULATOR

In accordance with the procedure, the following takes place:

1. Storage Register = - 55_8 = - 000 101 101

   Accumulator = +600_8 = + 110 000 000

2. Complementing the storage register sign results in its containing + 000 101 101

3. Comparing the accumulator and storage register signs reveals they are alike.

4. Add:
   ```
   000 101 101
   110 000 000
   -----------
   110 101 101 with bits Q - 26 all zeros.
   ```

5. Placing the addition result in the accumulator, it now contains + 110 101 101

6. Comparing the accumulator and storage register signs reveals they are alike.

7. Checking for a Q carry reveals none.

8. The final answer in the accumulator is + 110 101 101, which equals + 655_8.

Note the identical manner in which the two problems were handled. In each case, the arithmetic was addition. In each case, the sign of the subtrahend (storage register operand) was inverted. Subtraction of unlike signs becomes addition, and it is not significant whether the accumulator is the larger or smaller operand. Although not illustrated by the example, generation of a Q carry when dealing with unlike signs represents an overflow.

Assume the problem (- 247_8) - (- 133_8) where - 247_8 is the addressed operand and - 133_8 is the implied operand. The subtraction is accomplished as follows:

1. Storage register = - 247_8 = - 010 100 111

   Accumulator = - 133_8 = - 001 011 011

2. Complementing the storage register sign results in its containing + 010 100 111.

3. Comparing the accumulator and storage register signs reveals they are unlike.

4. Complementing the accumulator results in its containing - 110 100 100, with bits Q - 26. all ones.

5. Add: 010 100 111

   110 100 100

   001 001 011 with a 1 carry making bits Q - 26 all zeros and resulting in a Q carry.

6. Placing the addition result in the accumulator, it now contains - 001 001 011.

7. Comparing the accumulator and storage register signs reveals they are unlike.

8. Checking for a Q carry reveals one.

9. Adding a 1 to accumulator value field position 35 and inverting the sign results in a final answer of + 001 001 100 in the accumulator.

Repeating the problem, but with - 133_8 the addressed operand, the action is as follows:

1. Storage Register = - 133_8 = - 001 011 011

   Accumulator = - 247_8 = - 010 100 111

2. Complementing the storage register sign results in its containing + 001 011 011

3. Comparing the accumulator and storage register signs reveals they are un~~a~~like.

4. Complementing the accumulator results in its containing - ~~010~~ 101 011 000, with bits Q - 26 all ones.

5. Add:

```
        001  011  011
        101  011  000
       ---------------
        110  110  011  with Q - 26 all ones.
```

6. Placing the addition result in the accumulator, it now contains - 110 110 011~~.~~, with Q-26 all 1's.

7. Comparing the accumulator and storage register signs reveals they are un~~a~~like.

8. Checking for a Q carry reveals none.

9. Complementing the accumulator value field results in a final answer of -001 001 100 which equals - 114_8.

Note that the Q carry serves to indicate the accumulator was the smaller operand and that the present accumulator value field is in true form and 1 less than the correct answer, when dealing with operands having like signs. The absence of a Q carry, on the other hand, indicates the accumulator was the larger operand and the present accumulator value field is the correct answer in complement form.

404. 3 Multiplication.

Binary computers perform multiplication by repetitive addition and shifting. The process is similar to that used when performing binary multiplication using pencil and paper. The basic rule is to add and shift when a 1 is encountered in the multiplier, and to shift, without addition, when a zero is encountered.

Assume the problem is to multiply 15_8 by 5_8. On paper we would do the following :

$$\begin{array}{rcrl}
15_8 & = & 1101_2 & \text{(multiplicand)} \\
5_8 & = & 101_2 & \text{(multiplier)} \\
\hline
 & & 1101 & \text{(first multiply by 1)} \\
 & & 0000 & \text{(multiply by zero -- no add -- shift)} \\
 & & 1101 & \text{(second multiply by 1 -- shift and add)} \\
\hline
 & & 1000001 & = 101_8
\end{array}$$

Proof : $15_8 \times 5_8 = 13_{10} \times 5_{10} = 65_{10}$
$101_8 = 65_{10}$

Notice in the first step we had a 1 in the end position of the multiplier. This is the position scanned in a binary computer. With a one in this position the multiplicand is put down as the first partial product. The second step calls for a multiplication by 0. No addition takes place, but the zero, and its relative position in the multiplier, is noted by ~~kplacing~~ placing a zero one place left of the first partial product, followed by the number of zeros equivalent to the number of places in the multiplicand.

The third step calls for a~~n~~ multiplication by 1, hence a shift and addition~~s~~. Once again the multiplicand is shifted to the left -- in recognition of the relative position of the multiplier bit. With this third shift and addition the problem is complete. This same procedure is followed in most binary computers. Shifting moves the partial product into the high order bit positions of the AC and, at the same time, moves the bit in the multiplier into position for sensing.

Bring in sign

Binary computers use three registers to accomplish multiplication. The contents of the SR are the multiplicand, the contents of the M@ the multiplier and ~~thk~~ the partial product or sum is placed into the AC. The AC and MQ are shifted simultaneously so on the first shift bit 35 of the AC beco mes bit 1 of the MQ. Bit 35 of the MQ is lost and bit 34 becomes bit 35. The final answer will be conta ned in both the AC and MQ with the AC containing the most significant portion of the answer and the MQ the least significant.

Computers have no place to store partial products so each partial product is added to the contents of the AC and the answer gradually built up in this manner. Also, the computer cannot recognize when multiplication has been completed and will continue multiplying until told to stop. For this reason a shift counter (SC) is used. Usually, the shift counter is set to contain a count equal to the total number of positions in the multiplier. In the 7094 this count would be 43_8 or 35_{10}. When a multiplication is to be performed that does not use all positions of the MQ the variable length mode of multiplication is used. In this mode the programmer inserts the proper count into the instruction and the SC is set to this value. When the SC= 0 the operation is complete and the computer will go on to the next instruction.

Assume we are to perform the same problem (15_8 x 5_8), using a binary computer with five-position registers. At the start of the problem the registers would contain:

SC = 101 (5_{10})   SR = 01101   AC = 00000   MQ = 00101

Bit 5 of the MQ is sensed to determine if it contains a 1 or a 0. Because it is a 1 the contents of the SR are put into the AC. The AC and MQ are shifted right one place to align the registers for the next step. This puts bit 4 of the MQ into position 5 for sensing and also puts the least significant bit of the answer into MQ position 1. The registers now contain:

SC = 100   SR = 01101   AC = 00110   MQ = 1] 0010

The bracket around the first bit in the MQ indicates this bit is part of the partial product. Bit 5 of the MQ is again sensed to determine if it contains a 1 or a 0. Because a 0 is encountered, no addition takes place but the AC and MQ are shifted right one place. The registers now look like this

SC = 011 SR = 01101 AC = 00011 MQ = 01]001

The 1 in MQ position 5 requires an addition and a shift. The contents of the SR are added to the AC and the registers now contain:

SC = 010 SR = 01101 AC = 01000 MQ = 001]00

MQ positions 4 and 5 now contain the last two bits of the original multiplier -- both zeros. These zeros will result in shifting without addition and at the end of the problem the registers will contain:

SC = 0 SR = 01101 AC = 00010 MQ = 00001

The operation is halted because SC = 0. The answer is contained in both the AC and MQ which equal $0001000001_2 = 101_8 = 65_{10}$.

Most binary computers perform a multiplication by examining the low order position of the multiplier to determine whether that iteration of the multiply cycle is to be a multiplication by zero or one. If the contents of MQ 35 contains a one, a multiplication by one is indicated and the multiplicand (storage register) is added to any previous partial product in the AC. The contents of the AC and MQ are shifted right one place to align the new partial product, and to place the next low order position of the multiplier in MQ position 35. If zero had been detected in MQ 35, a multiplication by zero would have been indicated and accomplished by shifting the partial product and multiplier without adding the multiplicand.

Thus, by examining one position of the multiplier and adding and shifting for the proper number of iterations, the multiply is performed. Because one position is examined, each iteration performed may be considered as a multiplication by either zero or one. The result of each multiplication is added to the partial product which, in turn, is shifted to maintain the proper relationship between the partial product and the multiplicand.

If it were possible to look at two bits at a time and multiply according to the sum of the two bits, machine time could be halved -- doubling the number of add iterations that could be performed during a machine cycle.

Two bits could have four possible configurations -- 00, 01, 10, and 11. The 00 combination is a multiply by zero, the same as in looking at a single bit. However, since we are looking at two bits the partial product would have to be shifted two places, rather than one as outlined above.

The combination of 01 results in a multiply by one and the computer would add the multiplicand to any partial product already formed and shift right two places.

The combination of 10 is a binary multiply by two. Any binary number is multiplied by two when it is shifted left one place. For example, the number 101_2 (5_{10}) is doubled when it is shifted left one place to become 1010_2 (10_{10}). Decoding a 10 bit combination, then, would result in shifting the multiplicand left one place, adding it to any partial product in the AC and then shifting the AC right two places. For example, multiply 101_2 by 0010_2 ($5_{10} \times 2_{10} = 10_{10}$). At the start the registers would contain:

SC = 4 SR = 0101 AC = 0000 MQ = 0010

On the first iteration the last two bits of the MQ indicate a multiply by two. The contents of the SR are shifted left one place and put into the AC. At this point the AC contains 1010. Next the AC and MQ are shifted right two places and the registers now contain:

SC = 2 (after two shifts) SR = 0101 AC = 0010 MQ = 1000

Decoding the last two bits of the MQ indicate that a multiply by zero is required. Addition does not take place and at the end of the iteration the registers contain:

SC = 0 SR = 0101 AC = 0000 MQ = 1010

The answer, 1010_2 ($12_8 = 10_{10}$), is contained entirely within the MQ and a multiply by 2_{10} has been effected.

It is only the last configuration of the two bits, 11, that indicates a problem area. This is a multiply by 3_{10} and cannot be achieved by shifting. Any shifting in a binary machine affects the answer by factors of two which makes it impossible to use the shift technique when multiplying by an odd integer. However, because $011_2 = 100_2 - 1$, it is possible to multiply by four and then subtract one to effectively multiply by three. Assume the problem is to multiply 5_{10} by 3_{10}. The correct answer is 15_{10}, of course, and, using our four-bit registers, they would look like this at the beginning of the add iteration:

SC = 4 SR = 0101 AC = 0000 MQ = 0011

In this multiplication we are going to utilize two extra positions in the AC -- Q and P. While these are normally used for overflow they will be used this time to remember that a complement addition has been performed.

Decoding the last two bits of the MQ reveals that a multiply by 3_{10} is required. Because we are actually going to multiply by four and then subtract one, we will do the subtraction first. The SR is complemented and added to the contents of the AC. A 1 is added to the last position to make this a true subtraction (addition of the 2's complement). Following is the state of the AC at the end of the addition:

SR complemented	1010
1's to Q, P and 4	11 1
Sum	111011

The AC and MQ are shifted right two places and the registers now contain:

SC = 2   SR = 0101   AC =1110   MQ =1100

Decoding the last two bits of the MQ calls for a multiplication by zero. However, since we performed a complement addition we will, in fact, perform a multiply by 1. Because the AC has been shifted right two places this new 1 is in the binary four position ($100_2 = 4_{10}$). Another way of expressing this would be to say we added the two 1's that initially were in the MQ and the result was a carry (1 + 1 = 0 and 1 carry). The carry goes into the next high order position and becomes the 1 which will accomplish the multiply by four. The multiply by 1 results in the following addition:

AC	1110
SR	0101
	0011

Notice that a carry out of the AC resulted. Following shifting, the registers contain:

SC = 0   SR = 0101   AC = 0000   MQ = 1111

The correct answer is contained entirely in the MQ. 1111_2 is equal to 17_8 which equals 15_{10}.

To summarize -- the multiply by four and subtract one is accomplished by adding the 2's complement of the SR to the AC and bringing hot 1's into positions Q and P. The two 1's in the configuration 11 are added and the resulting carry placed in the next high order position to indicate a multiply by four. Had the decoder encountered another 11 configuration the carry would have rippled through and the multiplication to take place would, in effect, be a multiply by 16 (10000_2). The carry resulting from a 11 bit configuration will be propagated through the multiplier so long as 1's are encountered. At the first 0 a true add will be performed and this is the first real partial product to be placed in the AC.

Multiplication

The 7094 actually multiplies as outlined in the examples given above. The increased operating speed of the 7094 required a departure from the standards of single, repetitive additions and shifting. IBM design engineers have incorporated new techniques in the 7094 that enable the computer to sense two MQ bits at a time -- resulting in multiplication by 0, 1, 2 or 3 (equivalent to MQ bits 34 and 35 containing 00, 01, 10 or 11, respectively). Also, in the 7094, addition and shifting are accomplished simultaneously. Lines between the AC and AD are kept "hot" during the multiply cycle so adder inputs (sums) are immediately available in the AC. These advances in computer design make it possible for the 7094 to perform six additions every machine cycle, or an addition every two clock pulses, a total elapsed time of some 330 nanoseconds per addition.

The heart of this new multiply concept is the decoder (Systems 02.13.75.1) which examines the last two bits of the MQ and also pre-senses the next two bits (positions 32 and 33). Although the results of sensing bits 32 and 33 are delayed, the effect on the next add iteration is noted and the computer conditioned for this next add cycle. The following table lists the various multiplier triggers that may be set during an add iteration.

MULTIPLIER TRIGGERS

Definitions --

Hi indicates the high order position of the two bits of the multiplier in the decoder.

Lo indicates the low order position of the two bits of the multiplier in the decoder.

EXC denotes the Boolean exclusive OR condition.

S indicates the string bit and is used in remembering that a complement addition had been performed during the previous iteration. Setting the string bit causes the multiplier to appear to the decoder as though a 1 had been added to it.

The term "cycle begin" refers to the condition of the string bit (on or off) at the beginning of the add cycle.

The term "cycle end" refers to the condition of the string bit at the end of the add cycle.

	Hi	Lo	S (cycle begin)	S (cycle end)	Multiplier Trigger
1.	1	1	1	1	0X
2.	0	0	0	0	0X
3.	0	0	1	0	+1X
4.	0	1	0	0	+1X
5.	0	1	1	0	+2X
6.	1	0	0	0	+2X
7.	1	0	1	1	-1X
8.	1	1	0	1	-1X

Where:
0X = multiply by zero, SR to AD lines blocked.
+1X = multiply by one -- add multiplicand to partial product.
+2X = multiply by two -- shift multiplicand left and add to partial product.
-1X = multiply by three -- subtract multiplicand from partial product and set the string bit trigger.

The multiplier trigger set conditions are defined by the following equations:

1. $+1X = (\text{Lo EXC S})\ \overline{(\text{Hi})}$

2. $+2X = (\text{Hi EXC Lo})\ \overline{(\text{Lo EXC S})}$

3. $-1X = (\text{Lo EXC S})\ (\text{Hi})$

4. $S = (\text{Lo EXC S})\ (\text{Hi}) + \overline{(\text{L o EXC S})}\ \overline{(\text{Hi EXC Lo})}\ (\text{Hi})$

To demonstrate the practical application of this chart, assume the MQ contains 00010111 in the last eight positions. Positions 34 and 35 will be decoded first, yielding a 1, 1, configuration. The condition where Hi = 1, Lo = 1 and S = 0 (cycle begin) is fulfilled by line 8 on the chart. With these conditions we would set the multiplier trigger to -1X, a multiply by three. By definition this would cause the computer to subtract the multiplicand from the partial product and to set the string bit trigger. The string bit trigger will be set at the end of the cycle as indicated by the 1 in column S (cycle end). Notice that these conditions also fulfill the equation on line 3 of (Lo EXC S) (Hi). An exclusive OR exists between Lo and S because one is up (Lo = 1) and one is down (S = 0), and Hi = 1.

This first iteration is complete when the complement addition has been performed. The computer will remain in L time for the rest of the multiplication.

MQ bits 32 and 33 were sensed in the decoder during the process of their being shifted into MQ positions 33 and 34. Actually, decoding of these bits is complete before the shifting stops and the output is immediately available for the next iteration. From this point to the end of the multiply operation the computer will continue to look at 32 and 33 and decode them before they are shifted into MQ 34 and 35.

MQ bits 32 and 33 contain a 0, 1, configuration. Because the string bit is set from the previous iteration a 1 will be added to the multiplier. Decoding produces the sum of 0, 1, +1, or, 10. This configuration fulfills the conditions set forth by line 5 of the chart and a +2X multiplication is called for. This also satisfies the equation on line 2. The +2X requires that the multiplicand be shifted left one place and added to the partial product. The AC and MQ are then shifted right two places and the iteration is complete.

MQ positions 32 and 33 now contain a 0, 1, configuration. The string bit was not set during the previous iteration and the condition satisfies the requirements on line 4 of the chart (also equation on line 1). Decoding calls for a +1X multiplication and the multiplicand will be added to the partial product. Shifting again takes place and the iteration is complete.

The last two positions of the MQ contain 0, 0. This is a multiply by zero as indicated on line 2 of the chart. No addition takes place (SR to AD lines are blocked) and when the AC and MQ have been shifted right two places the multiplication is complete.

Whenever a complement addition is performed during a multiply sequence, "hot" ones are brought into AC positions Q and P to remember that the partial product is in complement form. This reminds the computer to put 1's in adder positions Q and P to account for the missing positions of the SR. (SR is in complement firm so use 1's). Recomplementing takes place on the first true add and the partial product will again be meaningful.

Decoder Network

The MQ decoder network is shown in Figure 404-1. The E and L callouts on the -AO blocks indicates the cycle in which the outputs are valid. Recall that a multiply instruction will result in I, E, and then the required number of L cycles to complete the operation. During the E cycle the original contents of MQ positions 34 and 35 are decoded. During subsequent L cycles, MQ positions 32 and 33 are decoded. Figure 404-2 shows the various triggers set as a result of decoding the contents of the MQ. Notice there is no 0X trigger and, in a sense, the 0X callout on the chart is a misnomer. Actually, there is no gating of the SR on a 0X so, in effect, gate SR to AD is blocked.

One other trigger is important to the multiply sequence. This is the multiply cycle trigger, shown in Figure 404-3.

Solution of Typical Problem

Computer sequence for solving the problem of multiplying 33_8 by 27_8 is shown in Figure 404-4. This chart is to be studied in conjunction with the flow diagram, Figure 404-5.

Notice in the second iteration on the problem chart that a one is put into AD Q from AC P. This is the reminder that a complement addition has been performed. It is vital to keep this string of one's in the high order bits so that when a true add is performed there will be a ripple out of the results of the complement addition.

Figure 404-6 is a print-out of an actual problem performed by the 7094. Notice that the shift counter is stepped two counts at a time. Notice also the setting of the multiply triggers as a result of decoding bits 32 and 33 of the MQ.

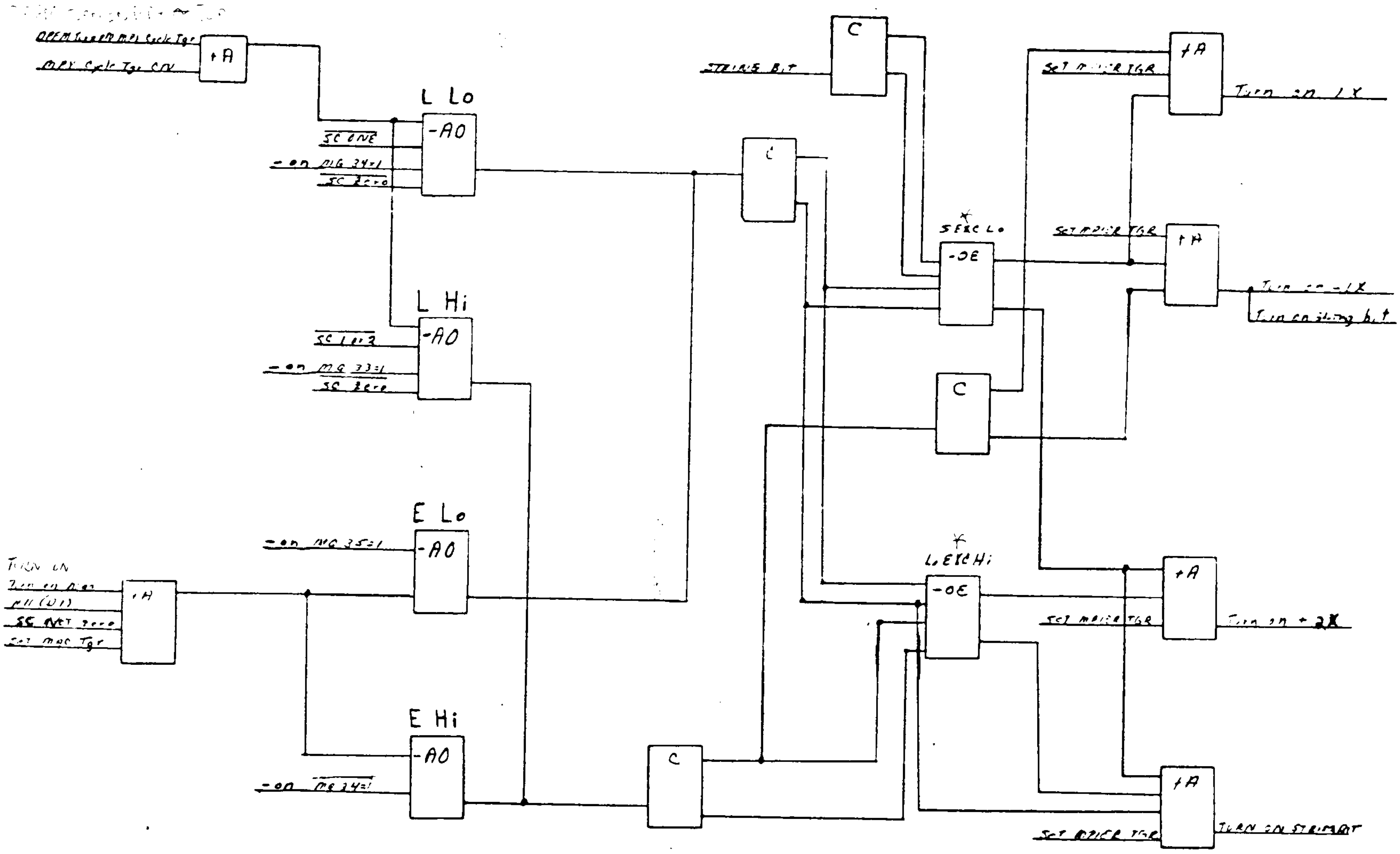


Figure 404-1 Decoder Network

page ~~404.27~~
404.26

Shift CTR	Storage Register	Adders	Accumulator	MQ	Iteration	S	MPY TGR	Remarks
	123456	QP123456	QP123456	123456				MPY $33_8 \times 27_8$
6	011011	100100 11 1 11100101	00000000 01111001	010111 010101	1	1	-1X	Complement SR → AD 1 → ADQ, P, 6 ADQ → ACP, sum to AC and shift right 2
4	011011	0110110 01111001 1 00101111	00001011	110101	2	0	2X	01 + S = 10 = 2X SR left 1 to AD AC to AD ACP to ADQ Sum to AC and shift right 2
2	011011	011011 00001011 100110	00001001	101101	3	0	1X	SR → AD AC → AD Sum to AC and shift right 2
0			001001101	101 = 1155				$33_8 = 27$ $27_8 = 23$ 621_{10} $1155_8 = 621_{10}$

Figure 404-4
7094 MPY Sequence

p1 c/3pp

MPY & VLM Flow Chart

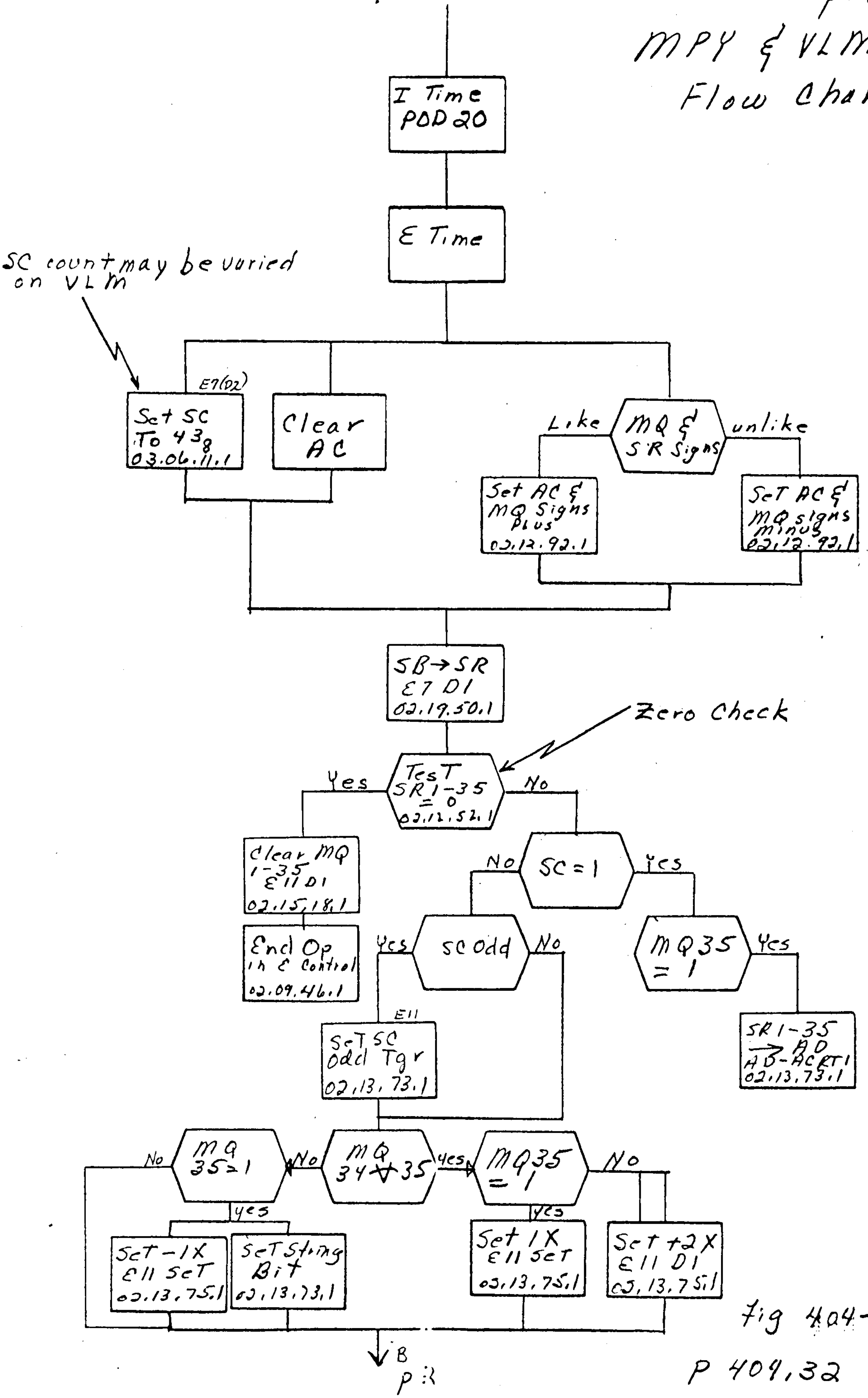

I Time
POD 20
E Time
SC count may be varied on VLM
E7(D2)
Set SC To 43_8 03.06.11.1
Clear AC
Like
MQ & SR Signs
unlike
Set AC & MQ Signs plus 02.12.92.1
SeT AC & MQ signs minus 02.12.92.1
SB→SR E7 D1 02.19.50.1
Zero Check
Yes
TesT SR 1-35 = 0 02.12.52.1
No
Clear MQ 1-35 E11 D1 02.15.18.1
End Op in E Control 02.09.46.1
No
SC = 1
Yes
Yes
SC odd
No
MQ 35 = 1
Yes
E11
SeT SC Odd Tgr 02.13.73.1
SR 1-35 → AD AD-AC RT1 02.13.73.1
No
MQ 35=1
No
MQ 34 ∀ 35
yes
MQ35 = 1
No
yes
yes
SeT -1X E11 SeT 02.13.75.1
SeT Sting Bit 02.13.73.1
Set 1X E11 SeT 02.13.75.1
SeT +2X E11 D1 02.13.75.1
B
p 3

fig 404-5

P 404.32

MPY & DLM (cont'd)

Note: SC stepped every clock pulse. Iteration begins at L0 & MQ SHIFTED RIGHT

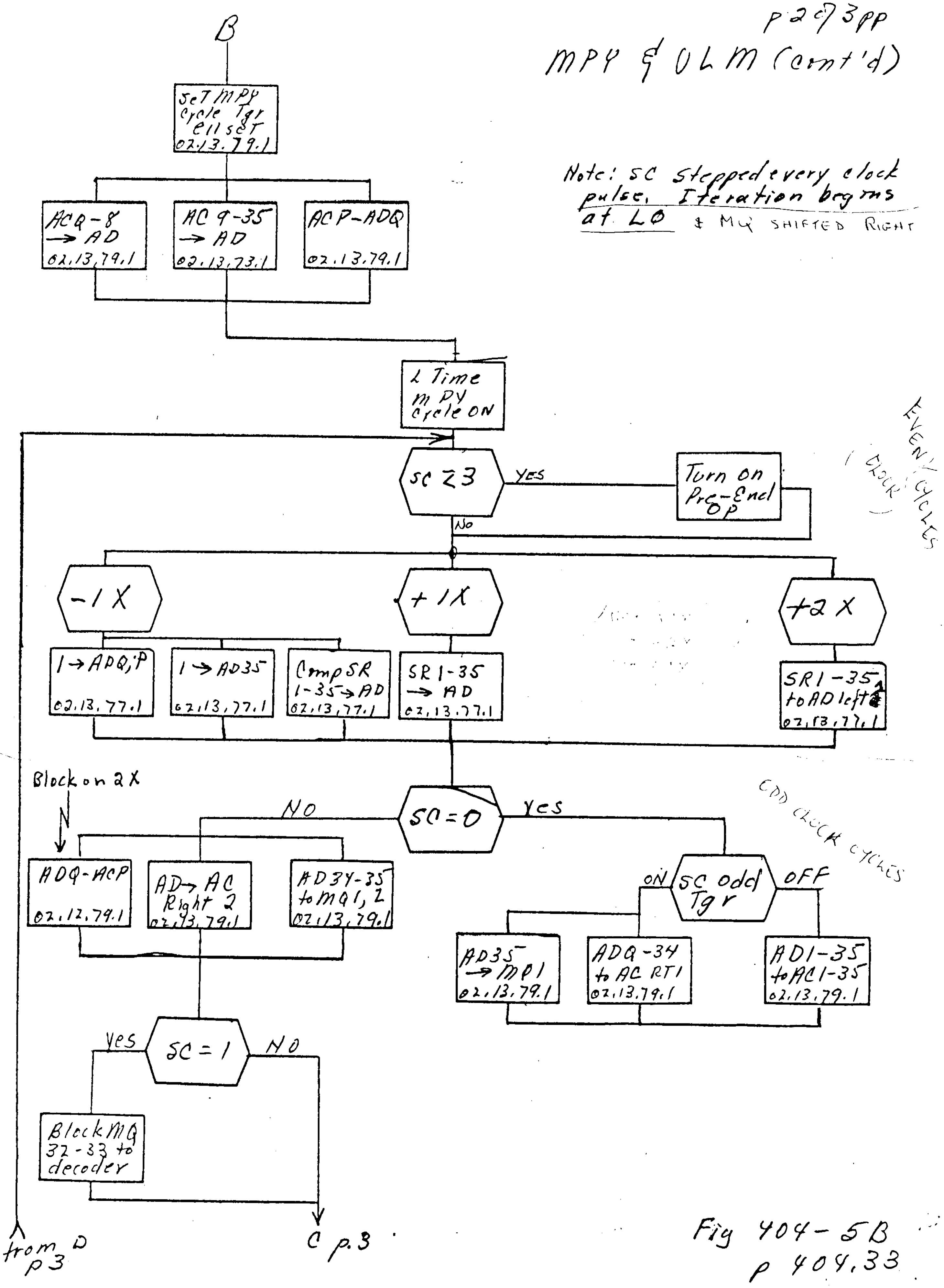
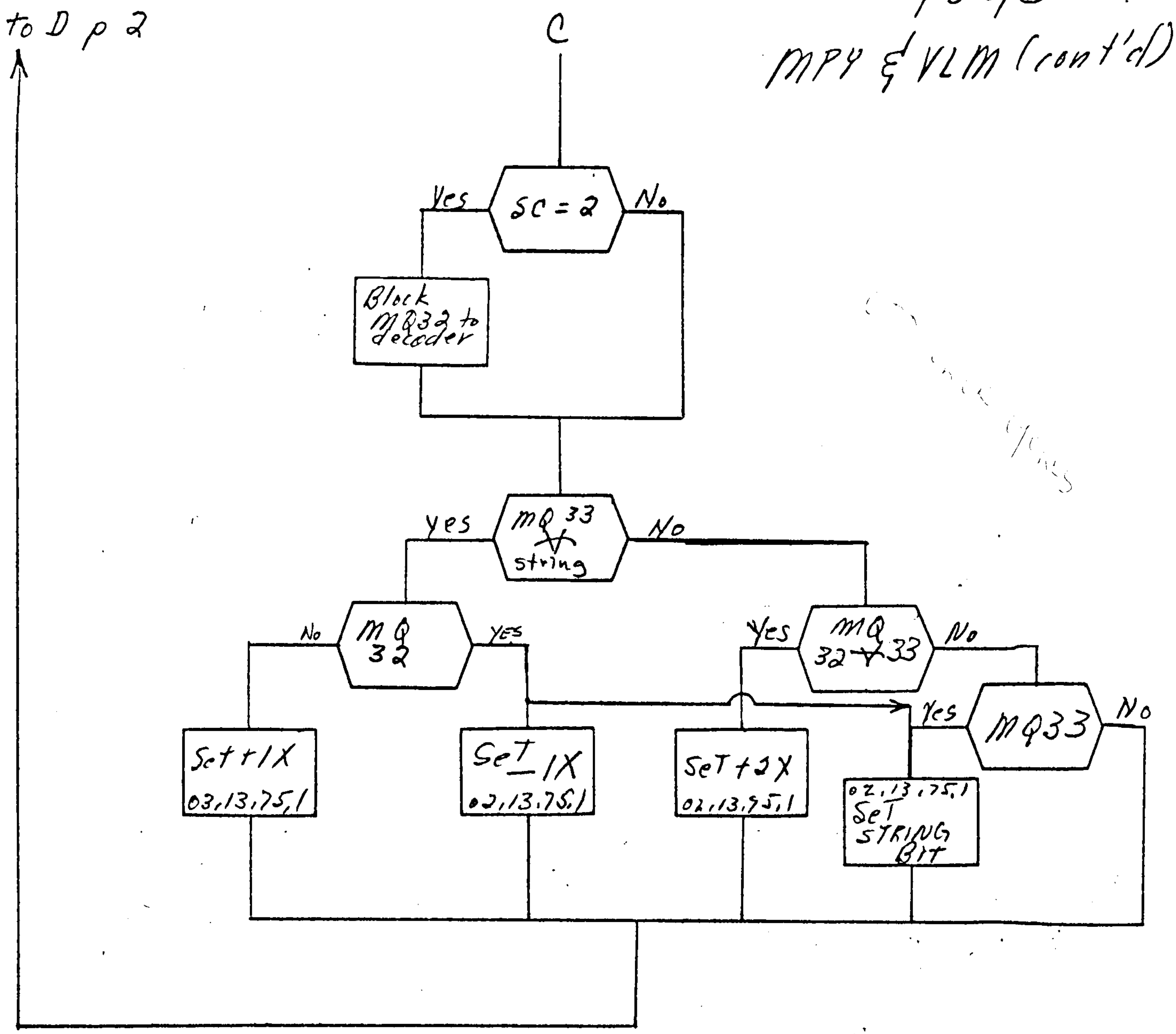


Fig 404-5B
p 404.33

p 3 of 3

MPY & VLM (cont'd)



404.33.1

Fig 404-5C

~~p 404.34~~

MPY WITH SR - 363535717237 MQ - 072736364777

NEXT	ITR	AC	S	Q	P	1-35	MQ	S	1-35	SC	STRING BIT	TIME
-1X			+	0	0	000000000000		+	072736364777	43	OFF	E
0X			+	0	1	303050414130		+	116567475177	41	ON	L
0X			+	0	1	360612103026		+	023535717237	37	ON	L
0X			+	0	1	374142420605		+	204727363647	35	ON	L
2X			+	0	1	377030504141		+	141165674751	33	ON	L
2X			+	0	0	171465070547		+	330235357172	31	OFF	L
2X			+	0	0	230174165651		+	166047273636	27	OFF	L
-1X			+	0	0	237716005071		+	335411656747	25	OFF	L
2X			+	0	1	353034015346		+	267302353571	23	ON	L
2X			+	0	0	164465753011		+	055660472736	21	OFF	L
-1X			+	0	0	226774342321		+	313354116567	17	OFF	L
2X			+	0	1	350647504614		+	262673023535	15	ON	L
-1X			+	0	0	164030670662		+	254556604727	13	OFF	L
2X			+	0	1	340056572304		+	353133541165	11	ON	L
1X			+	0	0	161672506200		+	727626730235	7	OFF	L
-1X			+	0	0	131306105307		+	356545566047	5	OFF	L
2X			+	0	1	331332035412		+	073531335411	3	ON	L
X			+	0	0	160145357022		+	016726267302	1	OFF	L
			+	0	0	070062567411		+	077353133541	0		L

Figure 404-6. 7094 MPY PRINTOUT

404.4 Fixed Point Division

Fixed point binary division in the 7094 is accomplished by dividing the contents of the AC and MQ, taken together as the dividend, by the contents of the SR, the divisor. A 35-position quotient is developed in the MQ with the remainder, if any, left in the AC. The sign of the MQ is set to the algebraic sign of the quotient, as determined by the SR and AC signs. The sign of the remainder remains the same as the sign of the dividend. The size of the registers restricts the size of the factors to be divided. The quotient can never exceed 35 bits, the maximum length of the MQ. If the AC portion of the dividend is equal to or greater than the divisor, the quotient would be too large for the MQ. In this case, division cannot take place and the computer is stopped with the divide check indicator on.

The following demonstrates binary division :

```
            0110  Quotient (MQ)
    1101 | 1000010  Dividend (AC and MQ)
           1011
          01011
           1011
          00000
           0000
            0000  Remainder (AC)
```

Notice that the divisor will go once or not at all into the high order positions of the dividend. Therefore, it is only necessary to determine if the divisor is equal to or smaller than these positions of the dividend. If the divisor is equal to or smaller than the selected positions of the dividend, a one is put into the quotient and the divisor is subtracted from that portion of the dividend. If the divisor is larger than the selected portion of the dividend, a zero remains in the quotient. Another position of the dividend is now taken into account, and the procedure starts again. This continues until all positions in the dividend have been tested.

In the 7094 the SR and AC are complement added to determine if a reduction of the high order positions of the dividend is possible. If a reduction is possible, these positions of the dividend are reduced by the amount of the divisor and the difference put into the AC. If a reduction is not possible the AC remains the same. A successful reduction results in a one being put into the MQ (position 35). Following the reduction attempt the AC and MQ are shifted one place left to bring the next position of the AC into alignment and another reduction attempted. This repetitive process continues until all positions of the MQ portion of the dividend have been moved to the AC. The SC will equal zero when the division is complete.

The following steps constitute a divide iteration in the 7094 :

1. Add [redacted] AC to SR and check for Q carry.

2. Complement bit position 1 of the MQ and shift AC and MQ left one place.

3. If a Q carry results, leave contents of the AC unchanged and put a zero in MQ position 35.

4. If no Q carry results, reduction has been accomplished. Put sum of reduction into AC and insert a 1 into MQ position 35.

A 7094 divide iteration takes two clock pulses so six iterations can be performed ~~in~~ during a machine cycle. The maximum number of cycles for any divide problem is eight, the minimum (variable length) is three.

Figure 404-7 is a flow chart of a DVH instruction sequence. During E time the contents of the AC are put i·to complement form and added to the

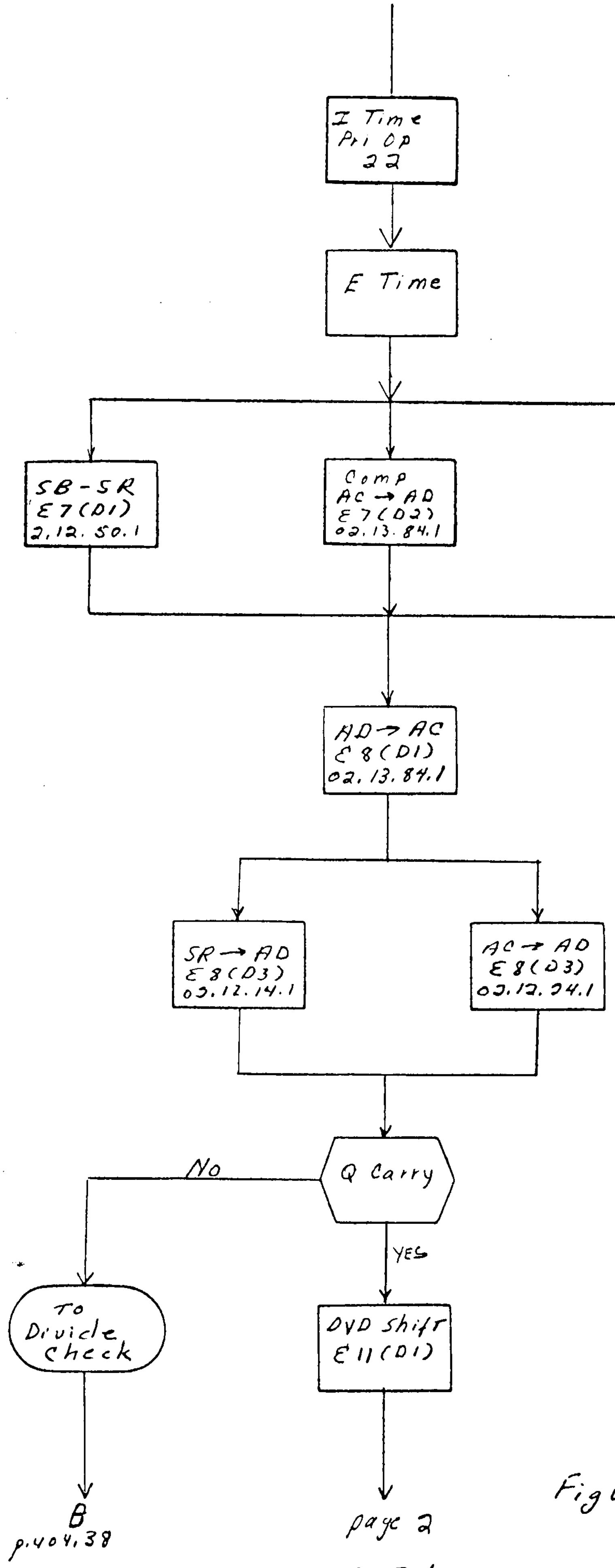


Figure 404-7A
DVH Flow Chart

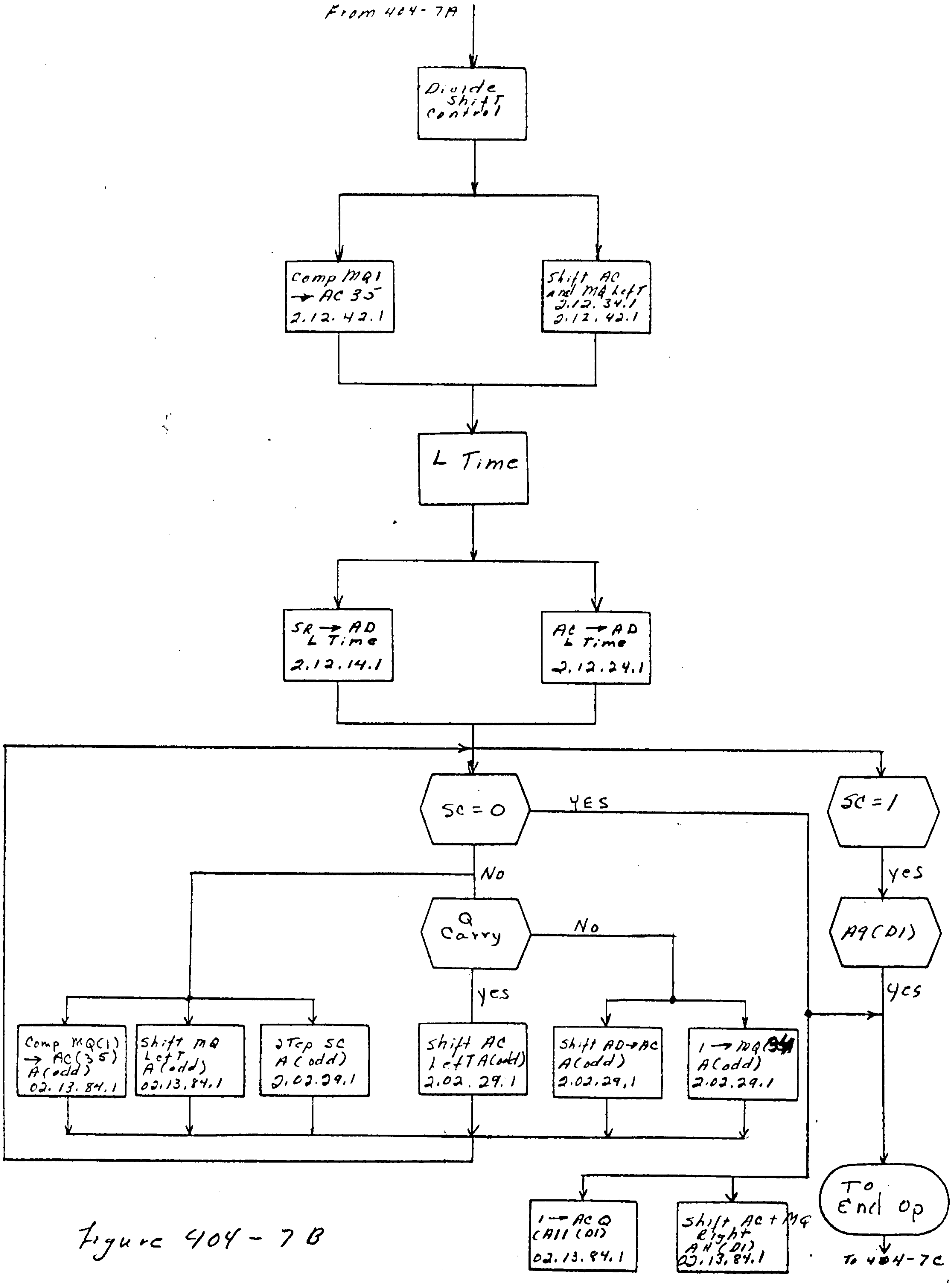


Figure 404-7B

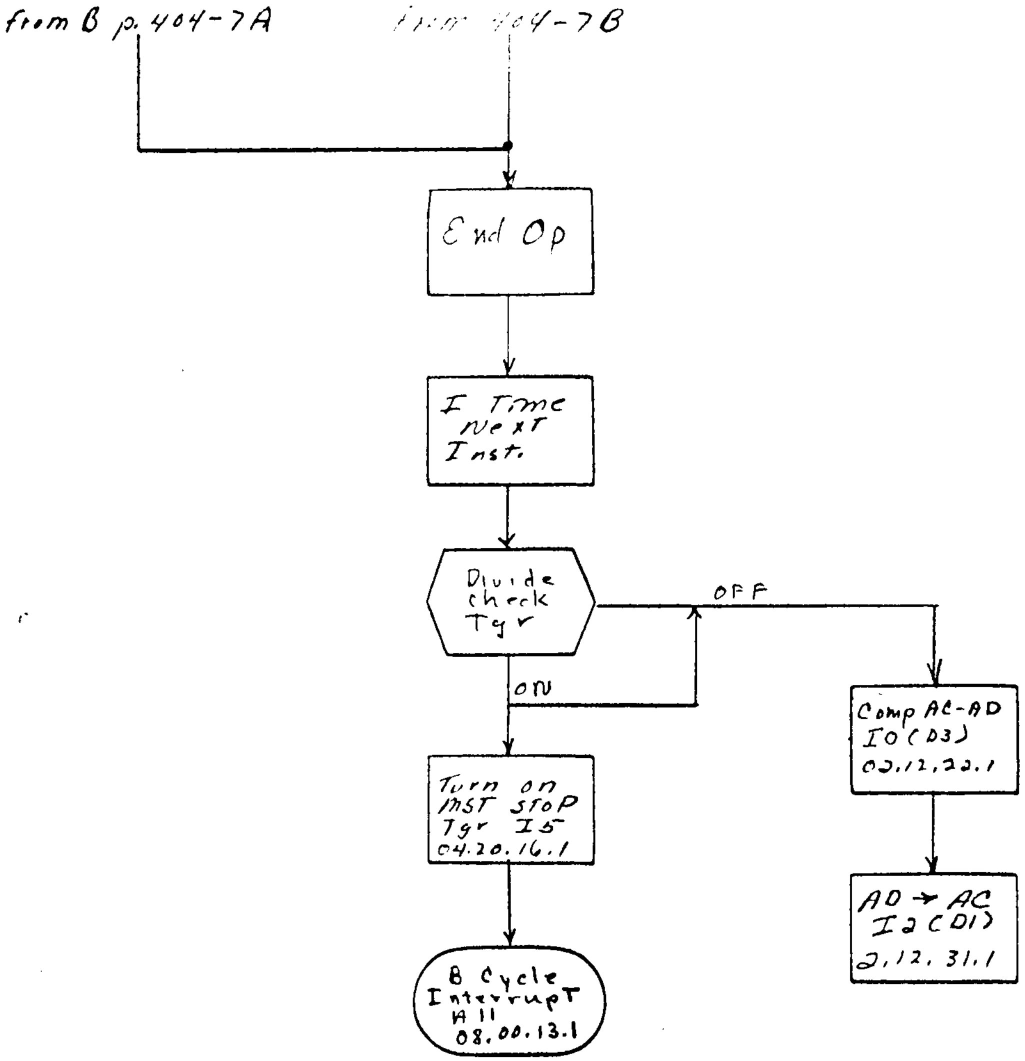

from B p. 404-7A
from 404-7B
End Op
I Time Next Inst.
Divide check Tgr
OFF
ON
Turn on INST STOP Tgr I5 04.20.16.1
B Cycle InterrupT All 08.00.13.1
Comp AC-AD I0 (D3) 02.12.22.1
AD → AC I2 (D1) 2.12.31.1

Figure 404-7C

contents of the SR. This first addition determines whether the quotient will be small enough for the MQ -- the result of the addition is not recorded. A Q carry indicates that division is possible. Lack of a Q carry indicates that division is not possible. Also during E time, the SC is set to 43_8.

When a Q carry does not result from the addition mentioned above, the divide check trigger is turned on and the computer signalled to divide check end operation. During the I cycle of the next instruction, the master stop trigger is turned on, causing a B cycle interrupt to be activated. The B cycle interrupt prevents I, E and L cycles.

A Q carry result from the E cycle test addition allows normal divide operation to take place. During the end of the E cycle MQ (1) is complemented and the combined AC and MQ are shifted left. This lines up the registers for the first reduction attempt and the computer goes into L time.

If the reduction is successful (no Q carry), a 1 is put into MQ 35 and the resultant sum of the reduction attempt shifted to the AC. MQ (1) is again complemented and the AC and MQ shifted left for the next iteration.

If the reduction is not successful (Q carry), a zero is entered into MQ 35, MQ (1) is complemented and the AC and MQ shifted left one position.

Notice from the flow chart that lines from the SR to the AD and from the AC to the AD are kept "hot" all during L time. Refer to the section on the adders (functional description) to see how fast the Q carry is felt during a divide operation.

The above sequence of divide iterations continues until SC = 0 when divide end operation is actuated.

During the I cycle of the next instruction the AC is again complemented to make the remainder a true number.

The following chart of an actual problem should be studied with reference to the flow chart. The problem is to divde 17_8 by 3_8. The registers have this configuration at the beginning of the operation :

SC = 6   SR = 000011   AC = 000000   MQ = 001111

Divide Check Test :

$\overline{AC}$ to AD   111111
SR to AD   000011
Q carry ←-000010

Turn on Divide Trigger

First Reduction Attempt

$\overline{AC}$ to AD   111111   MQ   001111
Complement MQ 1 and shift left

AC   111111
SR   000011
Q Carry← 000010 no reduction so 0 to MQ 6 and MQ now =011110

--------- End of first iteration and divide check ----------------

Second Iteration

SC = 5   SR = 000011   AC = 111111   MQ = 011110

Complement MQ 1 and shift left

AC   111111
SR   000011
Q carry ←--000010 no reduction so 0 to MQ 6, MQ = 111100

-------- End of second iteration --------------------------------

Third Iteration

SC = 4   SR = 000011   AC = 111111   MQ = 111100

Complement MQ 1 and shift left

AC   111110
SR   000011
Q carry ←----000001 no reduction so 0 to MQ 6, MQ = 111000

------------- End of third iteration ----------------------

Fourth iteration

SC = 3   SR = 000011   AC = 111110   MQ = 111000

Complement MQ 1 and shift left

AC   111100

SR   000011

NO Q carry ---111111 successful reduction so 1 to MQ 6, MQ = 110001

---------------End of fourth iteration ---------------------

Fifth Iteration

SC = 2   SR = 000011   AC = 111111   MQ = 110001

Complement MQ 1 and shift left

AC   111110

SR   000011

Q carry ---- 000001 no reduction so 0 to MQ 6, MQ = 100010

-------------- End of fifth iteration -----------------------

Sixth Iteration

SC = 1   SR = 000011   AC = 111110   MQ = 100010

Complement MQ 1 and shift left

AC   111100

SR   000011

NO Q carry ----111111 successful reduction so 1 to MQ 6, MQ = 000101

------------- End of sixth iteration -----------------------

SC = 0 so begin end op . The AC is complemented to the adders and returned to the AC so AC = 000000 at end of problem. The answer 000101 (=5_{10}), is in the MQ.

405. Variable Length Arithmetic

Variable length arithmetic is fixed point arithmetic with operands of a length other than 35 bits. Variable length operations include multiply and divide. The decrement of the instruction is used as the count field in variable length instructions. The count specified is usually a value less than 43_8.

If a count of 43_8 is used, the instruction is handled exactly as a fixed point instruction. A count of 60_8 causes an indirect address cycle because the count field (12 - 17) will OR with the 12 - 17 positions of the IA word.

Variable length instructions are used to conserve machine time. For example, if the contents of the MQ during an MPY instruction are less than 43_8, say, 30_8,-- it would be pointless to multiply by the last 11 zeros if it was not necessary. By setting the SC = 30_8, the computer will halt when the last meaningful bit has been used.

405.1 Variable Length Arithmetic Instructions

Variable length multiply	VLM +0204	(Min I, E) (Max I, E, 3L)	Figure 405- 2

This instruction operates the same as MPY except that the number of multiplier positions to be tested is specified by the count in the decrement portion of the instruction. The difference between MPY and VLM occurs during the E cycle when the shift counter is set. Usually, this count will be less than 43_8. A count of 60_8 or greater will cause an I/A cycle. The following illustration, Figure 405-1, shows the AC-MQ relationship before and after multiplication.

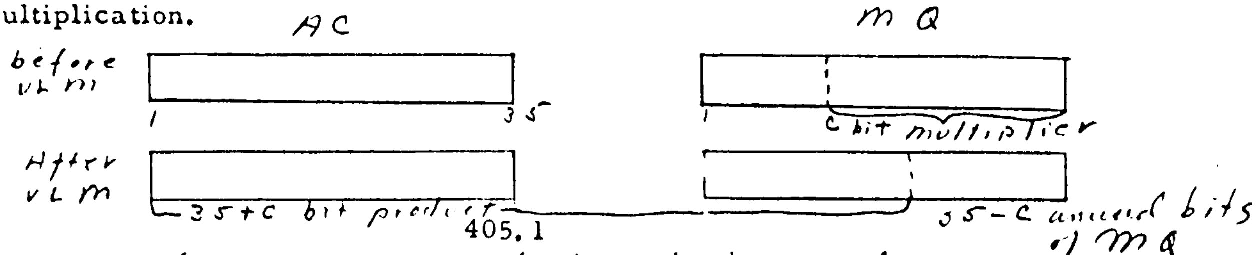


fig 405-1 Variable Length Multiply

VLM flow

fig 405 - 2

VDH + VDP

figure 405 - 3

405.2

Variable length divide or halt VDH (Min I, E) Figure 405-3
(Max I, E, 7 L)

This instruction operates the same as DVH, except that the number of reductions to be taken is specified by the count in positions (12 - 17) of the instruction. The number of positions in the quotient is equal to the count and will be contained in the low-order positions of the MQ. The count should be restricted to a number between 0 and 43_8. A zero count ends operation in E time and prevenst the shift at the end of the E cycle. A count of 43_8 will give the same result as DVH. A count greater than 43_8 causes part of the quotient to be shifted into the AC, where it can be altered. A count of 60_8 or greater will a cause an I/A cycle, and the count field (12-17) will OR with ~~the~~ 12 - 17 of the IA word.

Variable length divide or proceed VDP (Min I, E) Figure 405-3
(Max I, E, 7L)

The execution of this instruction is the same as VDH, except that the computer will not stop for a divide check, but will proceed to the next instruction.

FLOATING POINT ARITHMETIC

Arithemetic operations previously discussed used data in fixed binary point form, i.e., the binary point of the operands remained fixed throughout the calculations. When a large number of operands are required for a calculation, or the magnitudes of the operands are unpredictable and do not vary within known parameters, fixed point binary operations become exceedingly difficult. When such problems are encountered, an alternate mode of arithmetic is available in the 7094 - the mode of floating point arithmetic.

As the name implies, the binary point is not fixed during floating point operations; rather, it "floats", or is repositioned according to the operands involved in the calculatin. Computer action is much the same as that taken when calculating with paper and pencil. Floating point arithmetic instructions cause the computer to automatically position the operands used and to deliver the result in correct form,

The principle on which floating point arithmetic works is basic to mathematics and is known as scientific notation. Scientific notation is used when a numerical expression becomes unwieldy as the result of being extremely large or extremely small. Small quantities are ususlly expressed direct, such as saying that something is "four feet high" or weighs "one hundred pounds". Direct expression becomes impractical when stating the unit for static charge, for example, which is one coulomb and equal to 6,300,000,000,000,000,000 free electrons. This numerical value is not only cumbersome, it could easily be expressed incorrectly by inadvertently dropping one or more of the trailing zeros. To avoid direct expression of this quantity, a coulomb is usually defined as the unit of static charge present when 6.3 X 10^{18} free electrons are collected on a single body.

The expression 6.3×10^{18} denotes exactly the same value as the number written out with all the trailing zeros, but it is much easier to state and not so susceptible to error. As shown by the example, scientific notation is arrived at by taking the magnitude digits (coefficient) of a particular value and multiplying them by the radix of the number system being used, raised to a power (exponent) which will correctly position the decimal point, thereby accurately expressing the total magnitude involved.

Other examples of scientific notation include the velocity of light, expressed as 2.998×10^{8} meters per second, and the angstrom unit, expressed as 1×10^{-8} centimeters. All these notations, if multiplied by the indicated power of 10, will give the value commonly associated with the measurement of the given quantity.

If the significant digits of a value expressed by scientific notation are shifted so that the decimal point falls in a different place, the accuracy of the expression can still be maintained by a corresponding change in the power to which the radix is raised. For example, all of the notations below will yield exactly the same result if multiplied out:

2.998×10^{8}	$.2998 \times 10^{9}$
29.98×10^{7}	$.02998 \times 10^{10}$
299.8×10^{6}	$.002998 \times 10^{11}$
2998×10^{5}	$.0002998 \times 10^{12}$

It can be seen that for each shift left of the number (assuming that the decimal point stays in a fixed position) the power of 10 must be reduced by 1 to maintain the equality of the expression. Similarly, for every shift to the right, the value of the exponent is increased by one. Shifting the significant digits of a value back and forth and making the corresponding changes in the power of the radix can be utilized to perform addition or any other arithmetic function. For example, assume that it is desired to add the following expressions:

$$\begin{array}{r} 3.75 \times 10^{3} \\ + \; 445 \times 10^{2} \\ \hline \end{array}$$

Because the exponents of the radix terms differ, a direct addition cannot be performed. However, one of the terms can be shifted until the exponents are of the same value; then the significant digits may be added, and the radix term may be carried to the sum. If the first expression is shifted, the result is as shown below:

3.75×10^{3} shifted right one place 37.5×10^{2}

$$\begin{array}{l} \;37.5 \times 10^{2} \\ +445 \;\; \times 10^{2} \\ \hline \;482.5 \times 10^{2} \end{array}$$

Multiplying this notation out yields a result of 48,250, the same as would be obtained by adding the true value of each expression. If the second expression were shifted, the result would be:

445×10^2 shifted left one place 44.5×10^3

$$\begin{array}{r} 44.5 \times 10^3 \\ +\,3.75 \times 10^3 \\ \hline 48.25 \times 10^3 \end{array}$$

Multiplying 48.25×10^3 out also yields 48,250, the correct result. From this simple example, it can be seen that it is necessary only to make the exponents of the radices the same value by shifting the significant digits one way or the other and then performing the desired arithmetic operation.

The principle of scientific notation can be summarized by stating that it utilizes two factors to indicate the magnitude of a measured value. One factor is the radix raised to a power (either positive or negative), and the second factor consists of the significant digits of the value. Changing one of these factors requires a corresponding change in the other to maintain the validity of the expression. These same rules may be applied to the binary number system.

Notation with the Binary System

Because the binary system uses a radix of 2, all forms of scientific notation will be expressed in terms of powers of 2. In addition, since only symbols of 0 and 1 are employed in this system, the significant part of the notation will consist of a combination of 0's and 1's. For example, the value 256_{10} expressed in binary is 100 000 000 (400_8). If this binary number is considered to be an integer, scientific notation of the value would be as shown below:

$$100\ 000\ 000 \times 2^{\bullet}$$

Of course, this expression could be given in many forms, all of which are equal in value, by shifting the bits of the expression and changing the exponent of the radix 2. For example, all of the following expressions are equal to 100 000 000:

$$010\ 000\ 000 \times 2^{1}$$

$$000\ 001\ 000 \times 2^{5}$$

$$000\ 000\ 00.1 \times 2^{9}$$

The last expression has shifted the number so that it becomes a fraction, but no difficulty is encountered, since a binary fraction of this magnitude equals 5_{10}, or 1/2. The 9th power of 2 equals 512_{10}, and 1/2 x 512 yields a result of 256_{10}, the original value.

This type of notation is adequate for paper and pencil, but in a computer a different way of expressing the power of the radix is necessary. Since the 7094 is a binary machine, the power of the radix must also be expressed in binary. Thus, 2^{9} power will appear in some other form inside the machine, although the power indicated is still 2^{9}.

Floating-Point Data

It has already been stated that floating-point arithmetic in the 7094 uses operands expressed by scientific notation. Also, scientific notation has been defined as consisting of the significant digits of a value multiplied by the power of the radix. All that remains now is to show exactly how the power of the radix and the significant digits (or bits in binary) are expressed in the 7094. The format for a floating-point data word is as follows:

S	CHARACTERISTIC	FRACTION
	1 8	9 35

Bit positions 1-8 are referred to as the characteristic of the word, and they indicate the power of 2 to which the significant bits are raised. It can be assumed that 2 is the radix involved, since the binary system is being employed. If a characteristic of all zeros is arbitrarily chosen to represent 2^0, the range of exponents possible with 8 bit positions would be 2^0 - 2^{377}. However, this arrangement is impractical because it allows only positive exponents to be expressed, and it is desirable to express negative exponents as well. Therefore, the midpoint between the total number of exponents that can be expressed (400_8) has been arbitrarily chosen to represent 2^0. This value is 200_8. Thus, a positive power of 2 will be between the values 200_8 and 377_8, whereas a negative exponent will be between 0_8 and 177_8. For example, to express 2^9 as it is done in the machine, it is first necessary to change the exponent from decimal to octal form. Thus, 2^9 in the decimal system equals 2^{11} in the octal system. The radix is understood to be 2, so only the power (11) needs to be expressed. If 2^0 equals 200_8, then 211_8 equals 2^{11}. The actual appearance of the characteristic (in binary) which indicates 2^9 is as follows:

10 001 001

The characteristic part of the floating-point data word thus constitutes one of the two factors employed in scientific notation.

Bits 9-35 of the data word are called the fraction, and they constitute the significant bits of the value, or magnitude. The term fraction is used because the data contained in this part of the word is considered to be in fractional form, that is, a binary point is effectively located between bit positions 8 and 9, making all bits to the right of this point represent a value somewhere between -1 and +1. This fraction should not be confused with the fraction represented by a fixed point data word.. It is true that the numerical significance of these fractions is the same in that each position represents a power of 2, but the actual magnitude of the floating-point data word can be determined only after the fraction has been multiplied by the power of 2 indicated by the characteristic. In fixed-point data words, the magnitude of a number can be determined after the various powers of 2 present in a given word are added together.

The sign bit position of a floating-point data word represents the sign of the fraction; that is, if the sign bit is 0, the fraction portion is positive, and if the sign bit is 1, the fraction portion is negative.

A characteristic in the range 0_8 to 177_8 does not in itself indicate that the quantity is negative. To express a very small quantity may require a negative power of 2, but the quantity may still be positive. For example, to express the value of 1/8 in floating-point form requires a negative exponent (assuming that the fraction is 1/2). The smallest position exponent that can be expressed is 200_8 or 2^0, and multiplying this by 1/2 still yields a result of 1/2. To obtain the quantity 1/8, a characteristic of 176_8 is required when the fractional part of the data word is 1/2.

The characteristic of 176_8 represents 2^{-2}, and multiply this by 1/2 yields the desired quantity:

$$2^{-2} \times 1/2 - 1/2^2 \times 1/2 - 1/4 \times 1/2 = 1/8$$

Similarly, a value of -1/8 would be shown as follows:

S	CHARACTERISTIC	FRACTION
1	176	400000000_8

This value is known to be negative because the sign bit is a 1. Thus, it is the sign bit and only the sign bit which determines the polarity of the value expressed.

As an example of a floating-point data word, assume that it is desired to express the value 256_{10}. This value may be represented in octal by 400 or in binary by 100 000 000. This term may be expressed in scientific notation by 100 000 000 x 2^0 or 000 000 000.1 x 2^9. Taking the latter case, and placing 2^9 (2" in octal) into the characteristic bit positions yields a result of 211_8. The fraction remains as is, and the sign bit is cleared; so the floating-point form of 25610 is as follows:

S	CHARACTERISTIC	FRACTION
0	211	400000000

Arithmetic operations with floating-point data words are performed in much the same manner as the addition of terms in scientific notation. The characteristics are made the same by shifting one of the fractions and making the corresponding change in the value of the characteristic. The two fractions are then added (assuming addition is the operation called for), and the charactertistic is assigned to the sum. While it is true that a certain amount of "lining up" may be necessary before a floating-point operation may take place (the characteristics must be made equal), this process is performed automatically by the machine and is not the concern of the programmer. Also, the result of the operation will be a value which does not require further manipulation before another arithmetic operation can take place. Thus, the "floating" operation which occurs in floating-point arithmetic is really nothing more than an adjustment of the characteristic to keep the value being expressed in the proper order of magnitude.

Double Precision

When a fixed point fraction is changed to floating-point form, the resulting characteristic and fraction may exceed 35 bits. In this case additional hardware is available to accommodate the longer length fraction. The addressed operand is always placed in the storage register. When a double precision addressed operand is required, the low order fraction bits are housed in the IBR register which is associated only with the storage register. In the case of the implied operand, the accumulator and MQ register combine to house the double precision number. The MQ register is assigned a characteristic which is 27_{10} less than that of the accumulator because the fraction contained in the MQ register in bits 9-35 is displaced 27 positions to the right of the accumulator binary point - the point just to the left of accumulator bit 9. No characteristic is assigned to the IBR because it serves either to hold the low order addressed operand until it can be operated on or the partial result developed during an arithmetic process. However, the MQ register always reflects the result of an operation; a separate characteristic must be assigned because it is required to be very accurate in a floating-point operation, and these low order bits must be dealt with separately.

The double precision operand contained in the combined storage-IBR register is as follows:

S	CHARACTERISTIC	HIGH ORDER FRACTION		LOW ORDER FRACTION
	STORAGE	REGISTER		IBR

Bit S, the sign bit, is the sign of the entire fraction. When bit S is 0, the fraction is positive; when bit S is 1, the fraction is negative. The characteristic indicates the power of 2 to which the fraction is raised. In double precision, the fraction is 54 bits in length; storage register bits 9-35 on the high order fraction bits, and the IBR, which is only 27 bits long, forms the low order fraction. The accumulator - MQ register double precision operand is shown as follows:

S	CHARACTERISTIC N	HIGH ORDER FRACTION		▨	CHARACTERISTIC N-27	LOW ORDER FRACTION

The only difference between the accumulator - MQ register operand and the storage-IBR register operand is the existence of a characteristic for the low order fraction. Notice, the MQ register sign bit is not used.

Floating-Point Spill

During the execution of a floating-point operation, the resultant characteristic in either the accumulator or MQ register may exceed eight bit positions in length. The existence of such a condition means that machine capacity has been exceeded: machine capacity is exceeded when the exponent goes beyond 377_8 or below 0_8. When the characteristic goes beyond 377_8, a condition known as floating-point overflow is said to exist. Similarly, if the characteristic tries to go below 0_8, a condition known as floating point underflow exists. These conditions are referred to collectively as floating-point spill.

Overflow and underflow may occur in either the accumulator or the MQ register. Upon sensing the existence of either condition, the processing unit places the address of the instruction causing the condition plus one into bits 21-35 of location 00000. In addition, one of the bits 14-17 of location 00000 is set to record the cause of the spill.

Normalizing

When a floating-point data word is being dealt with, it may be in one of two forms: normalized or unnormalized. A normalized number is one that contains the binary point of the fraction just to the left of the most significant bit. Since the binary point of the fraction is considered to be just to the left of accumulator bit 9 in a floating-point data word, bit 9 must contain a significant bit if the number is to be in normalized form; that is, bit 9 must contain a 1. Therefore, the absolute magnitude of the fractional part of a floating-point data word must be greater than or equal to 1/2, but less than 1 if the number is in normalized form. If the most significant bit is not contained in bit 9, the number is said to be unnormalized. Normalizing eliminates leading zeros from a fraction.

At the completion of an arithmetic operation, the result may be in either normalized or unnormalized form. Certain instructions in the floating-point arithmetic class of the 7094 contain the option to normalize the result if so desired. When this is done, the fraction is shifted left until a significant bit is contained in the accumulator bit 9 position. However, to maintain the equality of the expression, the characteristic must be reduced by 1 for each shift to the left that occurs. As an example, assume the result of an arithmetic operation appeared in the combined accumulator-MQ register as shown:

$$0.\ 10\ 001\ 011\ 000\ 111\ \ldots\ldots\ldots\ldots\ 0_2$$

The first eight bits of the accumulator contain the characteristic, 213_8; bits 9-35 of the accumulator and 9-35 of the MQ register contain the fraction, 070000000000000000_8. This expression is in unnormalized form because the fraction contains leading zeros. To normalize the fraction, the fraction is shifted left three places, with the bits leaving accumulator bit 9 being lost. To maintain equality of the expression, the characteristic is reduced by three. The normalized number becomes:

$$0.210.700000000000000000_8$$

The value of the expression is maintained in both cases; however, leading zeros have been eliminated from the fraction in the normalized form. When the result of an arithmetic operation is to be normalized, the normalizing process takes place automatically after the final result has been computed. Normalization is specified by a positive sign (S bit is 0) in the floating-point instruction word.

At this point, it may seem desirable to always have results appear in the normalized form. This would seem true because, as leading zeros are shifted out of the fraction, low-order bits enter the accumulator from the

MQ register, thus increasing the accuracy of the answer. However, there are instances when it is desirable to perform an unnormalized operation. For example, if the values being dealt with contained very small characteristics (large negative powers of 2), a series of operations could cause accumulator underflow when normalizing takes place. If the magnitudes of the numbers are known to be very small, accumulator underflow may be avoided by leaving the answer in unnormalized form.

Consider the MQ register after a floating point operation. It may contain an expression whose characteristic is always 27_{10} less than the accumulator characteristic. To maintain the difference in characteristics between the low order MQ register fraction bits and the high order accumulator fraction bits, normalization is performed before the MQ register characteristic is computed.

Zero Fraction

A floating-point number having a zero fraction can be treated in a variety of ways because the significance of a zero fraction operand depends on the arithmetic process to be performed. In addition and subtraction a zero fraction operand just means that the fraction portion of the answer is identical with the non-zero fraction operand. The result of the arithmetic is meaningful. In the machine, a zero fraction operand has no effect on the operation; the arithmetic is performed, allowing normalization of the non-zero operand fraction if specified. Naturally, should both operands contain a zero fraction, the answer has no meaning and can never be normalized. The probability of such a situation, however, is almost nil.

In multiplication a zero fraction has a vastly different meaning and is therefore treated quite differently. In multiplication a zero fraction multiplier results in a product containing a zero fraction: anything times zero equals zero. Likewise, a zero raised to some power is still zero. It serves no purpose to perform the operation because the result will be meaningless. Also, a zero fraction can never be normalized. Consequently, in single precision multiplication a zero fraction multiplier causes the operation to be terminated and the characteristic portion of the accumulator and MQ registers, which receive the result, to be cleared. In double precision multiplication and multiplicand is checked for a zero fraction. Effectively, a multiplicand with a zero fraction has the same meaning as a multiplier with a zero fraction: the result fraction will be zero. Consequently, a zero multiplicand fraction in double precision multiplication causes the operation to be terminated and the accumulator and MQ register characteristic portions to be cleared. In addition, a better use of hardware is realized by checking the multiplier in one case and the multiplicand in the other. The end result is a shortening of the time necessary to accomplish floating-point multiplication.

When dealing with division, the divisor or the dividend could contain a zero fraction. Each case has a different meaning and is therefore treated differently. The treatment, however, applies to both single and double precision operations. If the divisor has a zero fraction, the quotient cannot be determined; a divide check condition results, and the operation is ended.

The dividend, however, remains unaltered in the case. When the dividend contains a zero fraction, the quotient will be zero. Since the quotient will have no meaning, the operation is ended. However, in this case, the associated characteristic positions of the accumulator and MQ registers, which hold the result of a division, are cleared.

The above discussion pertains only to zero fraction operands. There remains the condition of the result of a floating point operation containing a zero fraction. Since in multiplication and division the result is also influenced by zero fraction operands, these cases have already been covered. Only addition and subtraction, then, have not been discussed. In either of these operations, a zero fraction result causes the associated characteristic to be cleared and the operation terminated.

Arithmetic of Floating Point

Addition of floating-point numbers is done by adding the fractions of floating-point numbers which have equal characteristics. The characteristics are set equal preceding the addition by placing the number with the smallest characteristic in the AC. The fraction is then shifted right, and for each right shift one is added to the characteristic. When the characteristic of the AC equals the characteristic of the SR, shifting stops, and the fraction of the AC and SR are added. Bits shifted out of AC (35) enter MQ (9). The sum appears in the AC and forms the most significant part of the answer. The least significant part is the bits that were shifted into the MQ. The MQ characteristic is set 27_{10} less than the AC characteristic to complete an unnormalized floating add. If it were a normalizing instruction, a check would be made to see if a 1 were in AC position nine. If AC (9) does contain a 1, the operation would be complete; if not, the AC would shift left until a one did appear in position nine. Shifting increases the number, so to keep it the same, the characteristic is reduced by the number of left shifts taken. Floating-point subtraction works the same except that the fractions are subtracted.

Floating-point divide is accomplished by dividing the fraction of the dividend by the fraction of the divisor and subtracting the characteristics. During the subtraction of the characteristics, the 200_8 that is added to all exponents is lost. Therefore, before the answer is final, 200_8 must be added to the quotient characteristic.

Floating multiply is accomplished by multiplying the fraction in the SR by the fraction in the MQ. The exponents in multiply are added, so in a floating multiply, the computer adds the characteristics. Because 200_8 had been added to each exponent originally, 200_8 must be subtracted from the characteristic. The most significant part of the product is the AC and the least lignificant part in the MQ.

Sign control is as follows:

Multiplication and Division	Signs of factors alike; answer plus
	Signs of factors unlike; answer minus
Addition	Answer always has sign of the largest factor
Subtraction	After sign of SR is inverted, answer always has sign of the largest factor

Floating-point Arithmetic Timing

Because of the shifting, comparing and normalizing operations involved in floating-point arithmetic a fairly complex timing sequence has been designed. A three-stage tally counter and seven floating add control triggers accomplish the necessary time references. The tally counter is used during multiply and divide instructions to keep count of the L cycles required to complete the arithmetic functions, the floating add control triggers are used in all arithmetic operations.

The tally counter, Systems 02.10.21.1, is stepped by pulses developed as first step multiply, or first step divide; second step multiply and so on, up to a count of three, at which time the operation may be terminated or the counter reset to zero and counting continued. The tally counter is conditioned by the arithmetic functions being performed, i.e., the continuing demand for L cycles to complete an operation. Once the operation is concluded, the tally counter is reset to one.

The implication here is that the output of the tally counter is continued so long as additional L cycles are required by the arithmetic units. This is the case, as well, for the floating add control triggers (FACT) 1 through 7.

These triggers are constantly being set and reset at the end of odd clock pulses. Note in Figure 406-1 that the FACT trigger shown has a trigger immediately beneath it. The purpose of the additional trigger is to remember which of the FACT was on at the time of reset. These FACT will be reset, and continue to be set so long as they are needed.

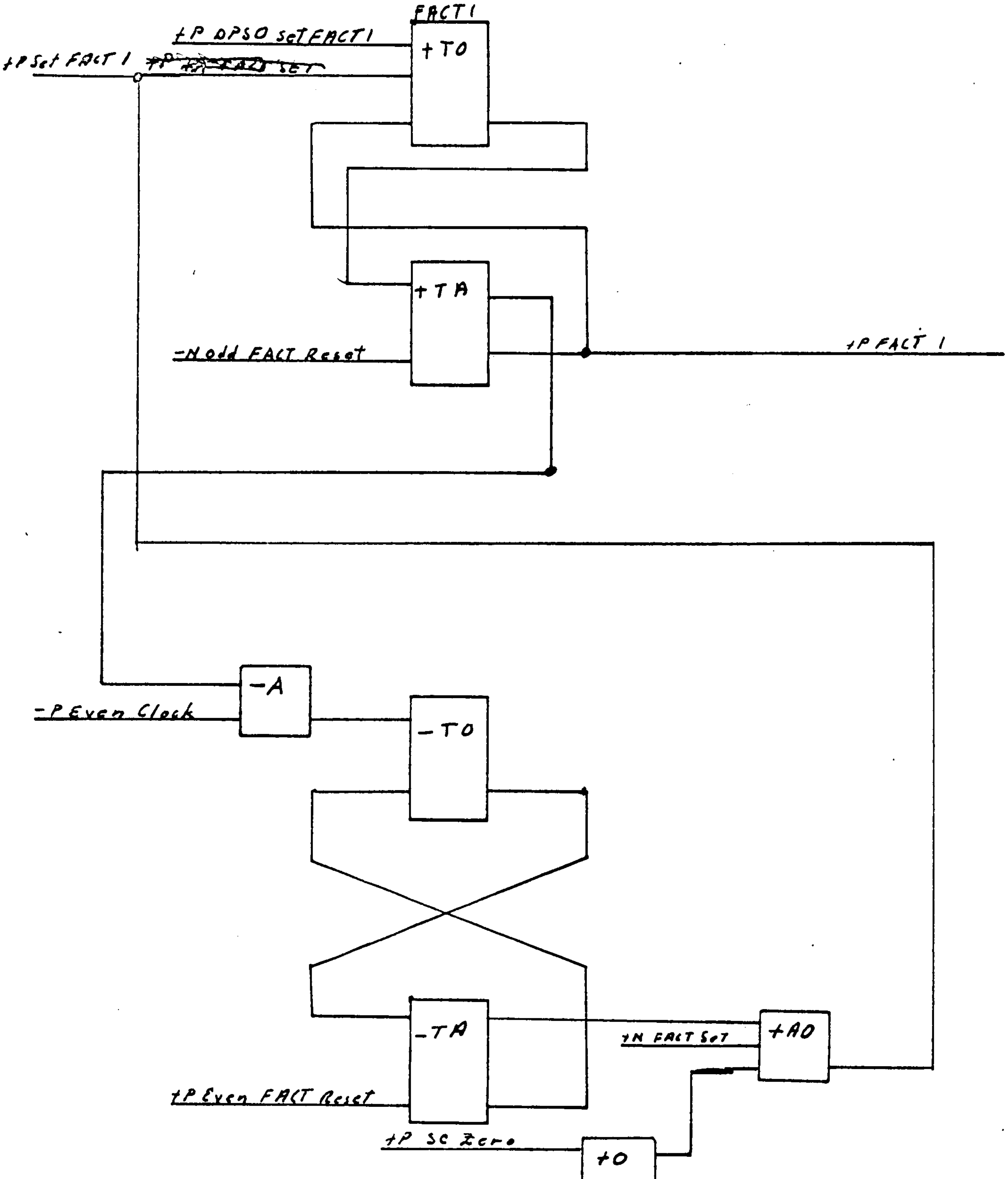


Fig. 406-1

~~406.5.1~~ 406,3.1

During the even and odd clock pulses following reset, the FACT control adder inputs and shifting. During the odd clock pulse, the FACT control adder outputs and new FACT settings. Each FACT except FACT 5 remains on only two clock pulses, although FACT 1 and 4 may be set repeatedly as they are needed during pre- and post-normalization. Following is a brief description of the FACT:

FACT 1: used during pre-normalization when the characteristics of the operands are made equal. (turns on also when characteristic equal but signs (unlike))

FACT 2: controls addition of the operands.

FACT 3: complements the AC or adds one to its contents depending on the previous 9 carry.

FACT 4: controls normalization, including adjustment of the characteristic.

FACT 5: controls fraction overflow shifting, MQ characteristic development and end operation.

FACT 6: controls complementing of the MQ fraction for those cases where the SR fraction was greater than the AC fraction, their data was not normalized to start, and significant data was shifted into the MQ and signs unlike.

FACT 7: controls placement of the MQ data corrected in FACT 6 back into the MQ.

Floating Add FAD + 0300 (Min 2 cycles) Figure 406-2
(Max 12 cycles)

This instruction algebraically adds :he floating-point number stored at the location specified by the address, t: the floating-point number in the accumulator. The most significant)art of the result appears as a normalized floating-point number in the accumulator. The least significant part of the result appears in the MQ as a floating-point number with a characteristic 27_{10} less than the characteristic of the number in the accumulator. The signs of the AC and MQ are set to the sign of the larger factor. If both the resulting MQ and AC fractions are zero, the registers will be reset to contain normal zeros with the signs corresponding to the original factor having the smaller characteristic. If the characteristics were equal, the resulting signs will correspond to the original AC sign.

The result in the AC is always normalized, whether the original factors were normal or not. No attempt is made to normalize the MQ.

In any floating point addition the first step taken by the computer is to equalize the characteristics. Assume the problem is to add 165_{10} and 1420_{10}. The correct sum is 1585_{10}. Upon conversion to binary these operands become:

$$165_{10} = 245_8 = 010\ 100\ 101_2$$

and

$$1585_{10} = 2614_8 = 010\ 110\ 001\ 100_2$$

As floating point arithmetic expressions these numbers become:

$$010\ 100\ 101 = .010\ 100\ 101 \cdot 2^9 = 1001.010\ 100\ 101$$

$$010\ 110\ 001\ 100 = .010\ 110\ 001\ 100 \cdot 2^{12} = 1100.010\ 110\ 001\ 100$$

Notice that the characteristic of the smaller number is 11_8 and that of the larger is 14_8. To equalize these two operands the smaller number must be shifted right three places and the characteristic increased by 3_{10}. The smaller operand then becomes:

$$1100.000\ 010\ 100\ 101$$

The two operands may now be added as follows:

$$\begin{array}{l} 1100.000\ 010\ 100\ 101 \\ \underline{1100.010\ 110\ 001\ 100} \\ 1100.011\ 000\ 110\ 001 \end{array}$$

This sum $= .011\ 000\ 110\ 001 \cdot 2^{12} = 011\ 000\ 110\ 001_2 = 3061_8 = 1585_{10}$

The above example illustrates a basic addition of two floating point numbers. Actual machine operation is contained in the following pages. While the floating add instruction is a simple one and basic to the 7094 arithmetic operations, complications do arise because of unlike signs, fraction carries and shifting significant bits into the MQ during equalizing. Figure 406-2 illustrates a basic FAD instruction. The problem has been specifically selected to illustrate one of the shortest routes to completion of a floating-add instruction. The letters in the lower left corner of each step in the flow diagram refer to the explanation below. At the start it is assumed the number in the storage register is 231607000000_8. The number in the accumulator is 231370000000_8. Recall that the adders contain the positions 9Q and 9P and the accumulator contains 9P. None of these positions exists in the storage register.

The following sequence of events takes place for the instruction FAD with the numbers specified above.

A. E7 time the primary operation code has been decoded as 30.

B. The C (SB) are gated into the SR. The various registers now contain (in positions S through 17):

	S	1	2	3	4	5	6	7	8	.	9Q	9P	9	10	11	12	13	14	15	16	17
SR	0	1	0	0	1	1	0	0	1				1	1	0	0	0	0	1	1	1
AD	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0
AC	0	1	0	0	1	1	0	0	1			0	0	1	1	1	1	1	0	0	0

C. A 1 is injected into position AD 9P, this will later develop a 9P carry if the sum of the fractions causes a carry into the characteristic.

D. A 1 is injected into position AD 8 to furnish the 1 needed for the two's complement of the accumulator. AC positions Q - 8 are complemented and gated to the adders at the same time the contents of the storage register are gated to the adders. The registers now contain (sum in adders):

	S	Q	P	1	2	3	4	5	6	7	8.	9Q	9P	9	10	11	12	13	14	15	16	17
SR	0			1	0	0	1	1	0	0	1			1	1	0	0	0	0	1	1	1
AD	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1
AC	0	0	0	1	0	0	1	1	0	0	1	0	0	1	1	1	1	1	0	0	0	

(the carry out of the adders is lost)

E & F. AC 9 = 0 but SR 9 = 1 so the normalized data latch (a reference for later shifting) is not set.

G. The characteristic difference of zero is checked by sensing the exclusive OR outputs of adder positions 1 - 8 (see Systems 02.13.47). Note that if a characteristic difference exists the contents of adders 3 - 8 are transferred to the shift counter.

H. Sign of the AC and the SR are checked and found to be alike.

I. SR 9 = 1 as we had previously determined.

J. FACT 2 is set as a result of the check made at point G. FACT 2 controls addition. The end operation trigger is also set and addition takes place during the next two clock pulses.

K. The C(SR 1 - 35) are once again brought to the adder input. Recall that previously we had brought the C (SR) to the adders to check for the characteristic difference between the operands. We are now ready for the actual addition to take place.

L. Again signs are checked and found to be alike. This seems repetitive but as will be shown on the master flow sheet for floating-point arithmetic, each time the signs are checked, more than one possibility of subsequent action exists.

M. The C(AC 9 - 35) are gated to the adders and addition of the C(SR) and C(AC) takes place. The carry developed from bit position 10 will cause a carry into 9, therefore a carry into 9P. Because we injected a 1 into AD 9P in step C, the carry is continued into AD position 8.

N. Positions AD 9 and AD 10 are checked and found to be 0.

O. The check made in step N causes the pre-post shift trigger to be set. The pre-post shift trigger will control shifting if normalization is required.

P. Signs are alike.

Q. The C (AD Q - 35) are shifted to the accumulator and AD 9 Q is shifted to 9P. The accumulator now contains:

S	Q	P	1	2	3	4	5	6	7	8	.	9P	9	10	11	12	13	14	15	16	17
0	0	0	1	0	0	1	1	0	0	1		1	0	0	1	1	1	1	1	1	1

Except for shifting due to the carry, this is the final answer.

R. Adder position 9P is sensed for a 1 and, since it does contain a 1, FACT 5 is set. We are not in L time so steps S and T are bypassed.

U & V. FACT 5 is set at the next odd clock pulse to control shifting of the AC -- as a result of the 1 in position 9P.

W. AC positions 9 - 35 are shifted right one place. At the same time the contents of AC 1 - 8 are increased by 1, to compensate for shifting the fraction.

X. A zero check is made to determine whether we have a true answer.

Y. The C (AC) is not zero.

Z, AA, BB. These operations would set the accumulator characteristic, less 27_{10}, into the MQ.

CC. FACT 5 is reset at 15 time and the end operation trigger is turned off. The answer to the problem is in the accumulator in normalized form.

The foregoing description of a FAD instruction covers one of the least-complicated examples of this instruction. It is possible that many other decisions and actions might take place during a FAD instruction involving other numbers. Figure is a flow chart showing all the possible contingencies that might develop during a FAD instruction.

sheet 1

FAD 0300

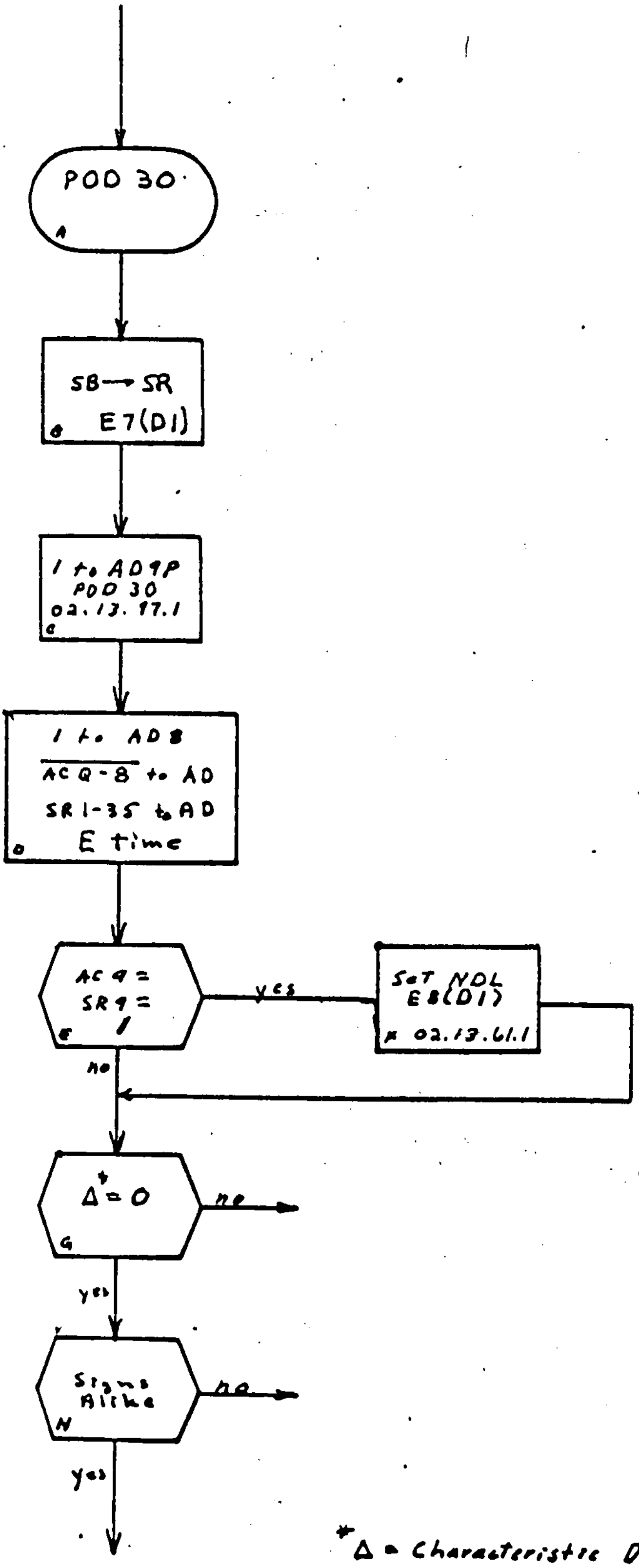


FIG 406-2

To Sheet 2

*Δ = Characteristic Difference

Sheet 2

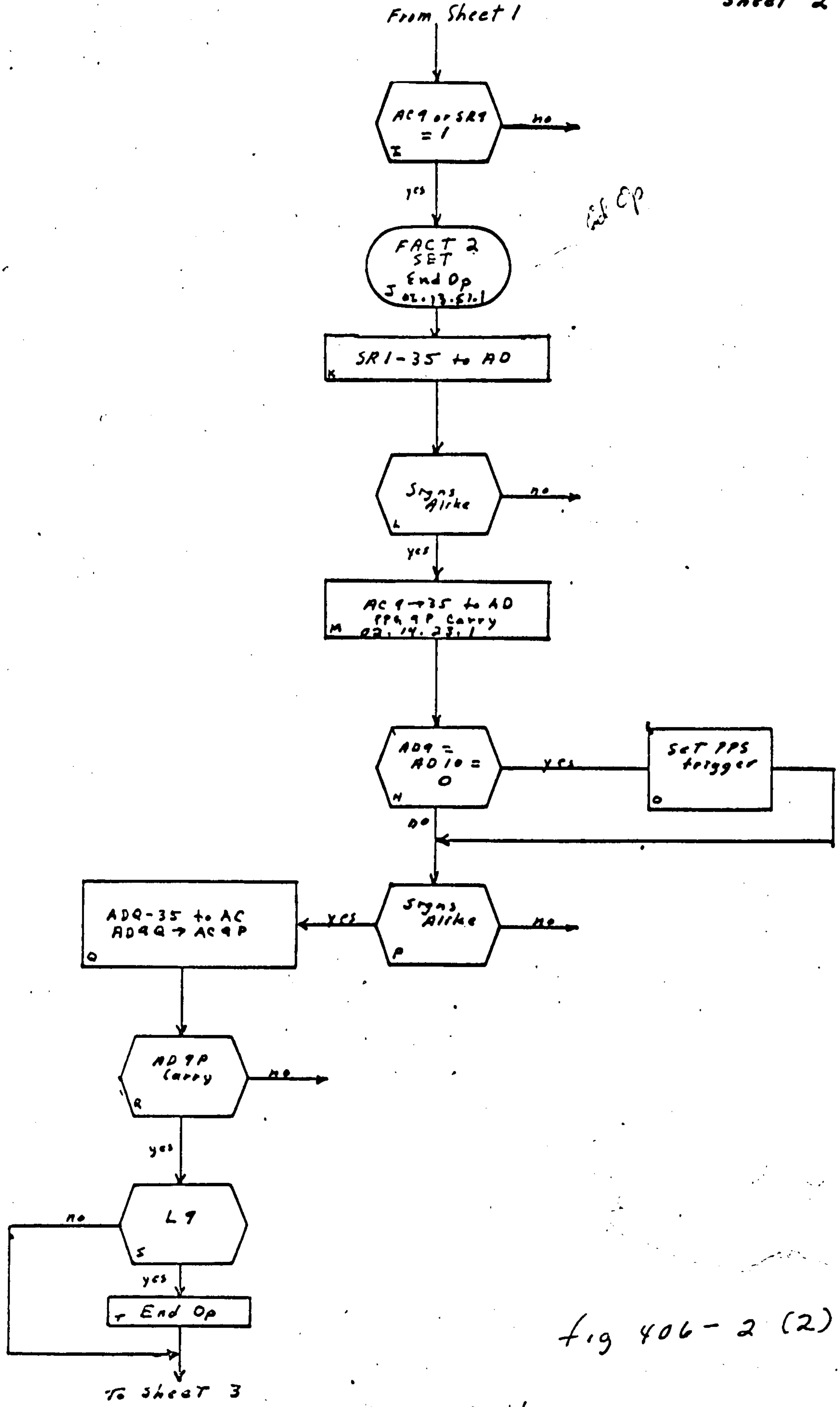


fig 406-2 (2)

sheet 3

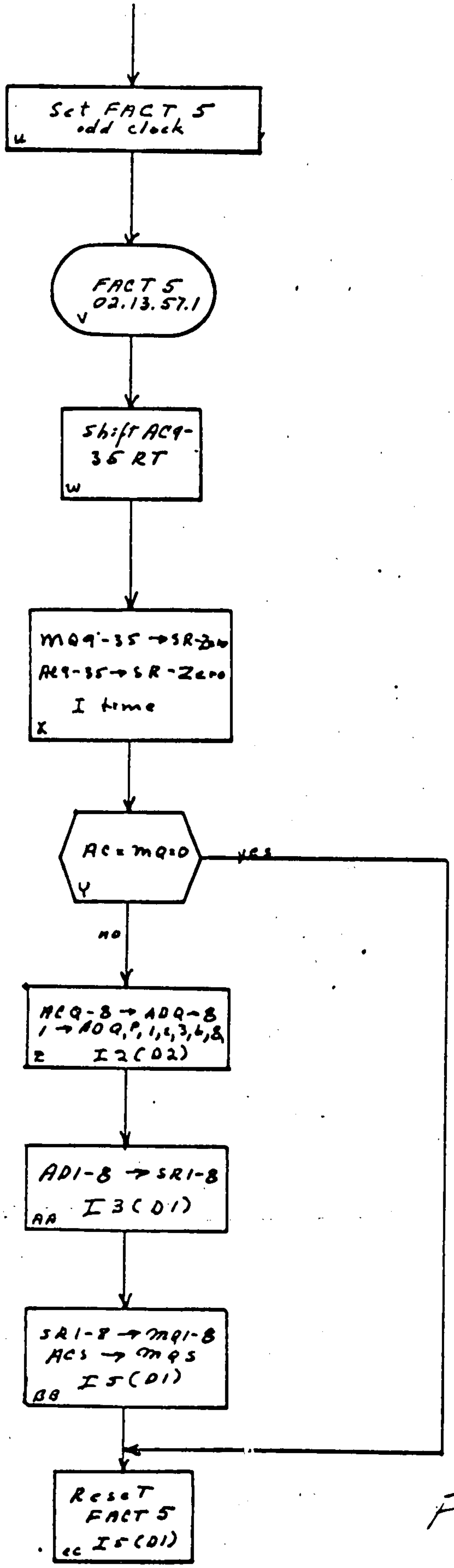


Fig 406-2(3)

7094 SINGLE PRECISION FLOATING POINT ADD

This instruction is asynchronous in operation and is controlled by I time, E time and 7 floating add control triggers (FACT 1 - 7). These FACT are always reset and set the latter part of each odd clock pulse. They serve to control adder inputs and shifting during the following even and odd clock pulses, adder outputs and new FACT setting during the odd clock pulses. Each FACT except FACT 5 will remain on only two clock pulses; however, FACT 1 and 4 may be set repeatedly as they are for pre and post normalization. FACT 5 will be reset only at I5 time. The following is a brief description of the functions of each of these controls.

E Time

E time is used for sign mixing, determination of characteristic difference and register swapping if necessary to place the smaller operand in the accumulator and the larger in the storage register.

If both operands have normalized fractions the normalized data latch (NDL) is turned on for reference during subsequent FACTS. If the characteristic difference is greater than 77_8 and larger of the two operands is normalized, it is placed in the accumulator, FACT 5 is set, and the operation is terminated. If the characteristic difference is equal to zero, either of the operands is normalized and the signs are alike, FACT 2 is set for addition and the operation is terminated. In all other cases FACT 1 is set for pre-normalization and the characteristic difference is placed in the Shift Counter.

The following extended description of floating add control triggers is to be used in conjunction with the flow chart to determine computer activity at any stage of the flow chart.

Floating Add Control Triggers

FACT 1

FACT 1 is used for pre-normalization. The contents of ACC (9 - 35) and MQ (9 - 35) are shifted right each clock pulse until the shift counter goes to zero. ACC 35 shifting to MQ 9 is determined by the NDL.

If the NDL is not on ACC 35 is shifted to MQ 9. If it is on and the signs are alike, a bit in ACC 35 will set the pre-post shift trigger (PPS) and ACC 35 will be complement shifted to MQ 9 after this first bit has been shifted, thus placing the two's complement of the value shifted out of the ACC into the MQ. This will save recomplementing the MQ after the addition. FACT 1 is set repeatedly until the shift counter is zero or one on an odd clock pulse at which time FACT 2 is set and shifting is completed. If this occurs at L 9 time (9 or 10 pre-shifts) and the NDL is on or an unnormalized instruction is being processed, the operation will be terminated.

FACT 2

FACT 2 controls the adding of the operands as follows:

TRUE ADD

If an overflow results from the add it is allowed to ripple into adder eight position to increment the exponent, the contents of the adders 9 - 34 are placed in the ACC and FACT 5 is set. FACT 5 is also set if no overflow occurs but adder 9 sum is one. The instruction is terminated for either of these cases or for an unnormalized instruction if the odd clock pulse is at L 9 time. FACT 4 is set if there is no overflow and adder 9 sum is zero.

COMP ADD

If no AD 9 CARRY results from the addition, FACT 3 is set for recomplementing the ACC. FACT 3 is also set if there is an AD 9 CARRY and the MQ is zero; this is done to complete the two's complement of the ACC by adding one to its contents.

FACT 6 will be set if there is an overflow, the contents of the MQ are not zero, and the NDL is off. This is for complementing the MQ. FACT 5 will be set if the NDL is on and normalization is not required, or FACT 4 set if normalization is required.

In either true or complement add the PPS trigger is set if AD 9 and AD 10 are zero.

FACT 3

FACT 3 is used to complement the ACC or add one to its contents depending on the previous 9 carry. FACT 4 is set if normalization is required or FACT 5 if normalization is not required. If this occurs at L 9 time the operation will be terminated if the NDL is on or the instruction is unnormalized.

FACT 4

FACT 4 controls post-normalization. The contents of the ACC and MQ are shifted left each clock pulse with MQ 9 shifting to ACC 35 until ACC 9 contains a one. If L 9 time occurs during FACT 4 and the ACC contents indicate less than six shifts are required the operation is terminated. FACT 4 is also used to adjust the characteristic by an amount equal to the number of shifts. This is done by subtracting 1 or 2 from the characteristic each FACT 4 depending on the PPS trigger which was initially conditioned in FACTS 2 or 3. If this trigger is on, it indicates that AC 9 and 10 are both zero and the characteristic must be decreased by a factor of two. The PPS is reset eachodd clock pulse of FACT 4 and set again if AC 10 and 11 are both zero.

FACT 5

FACT 5 controls fraction overflow shifting, MQ characteristic development and end operation. Existing 7090 trap checks and procedures are used during this operation.

FACT 6

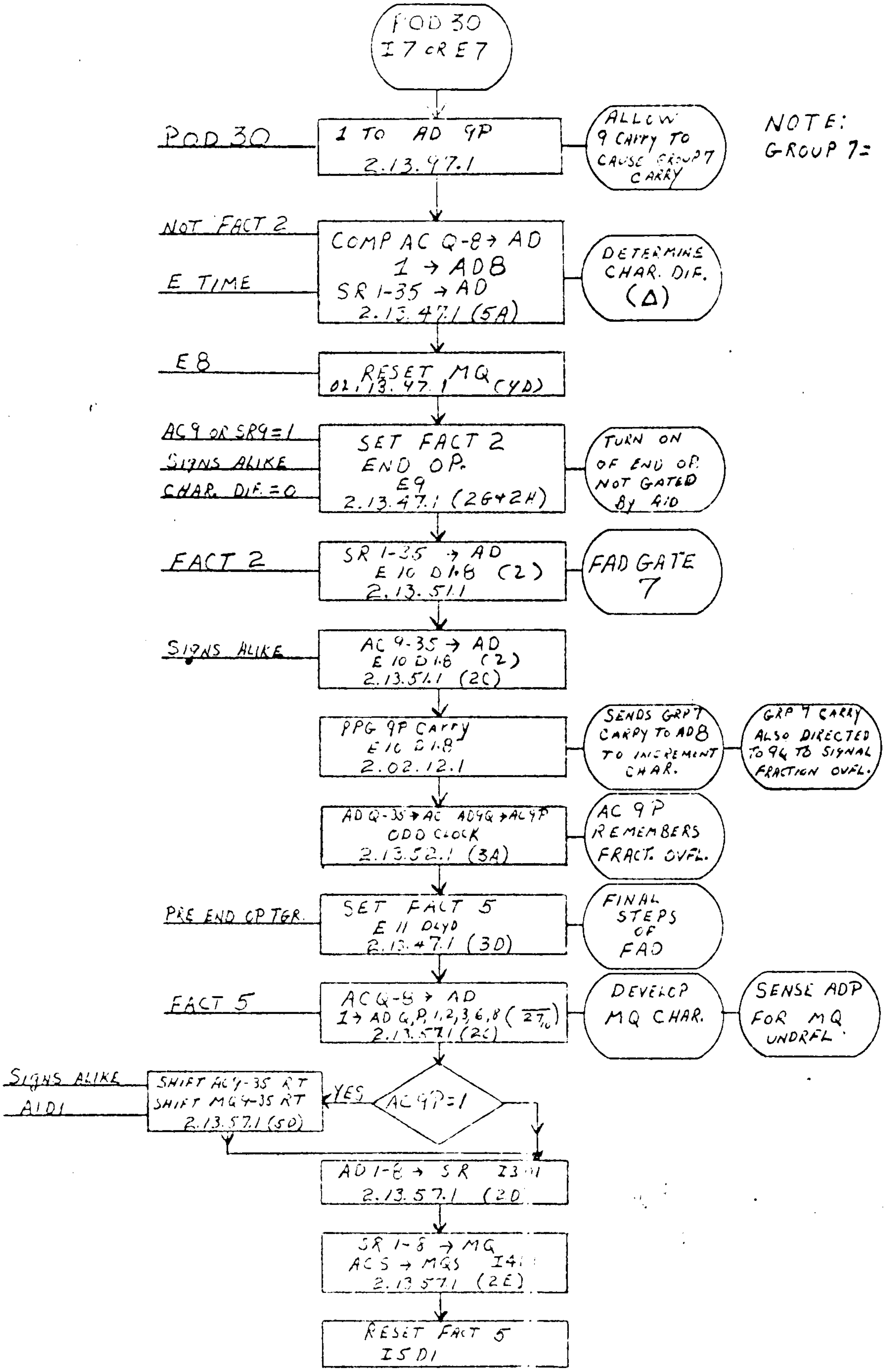
FACT 6 controls complementing of the MQ fraction for the cases where the SR fraction was greater than the ACC fraction, their data was not normalized to start, the significant data was shifted into the MQ.

FACT 7

FACT 7 controls placement of the MQ data corrected in FACT 6 back into the MQ. FACT 4 or FACT 5 is then set depending on normalization requirements.

FAD – FIRST CASE

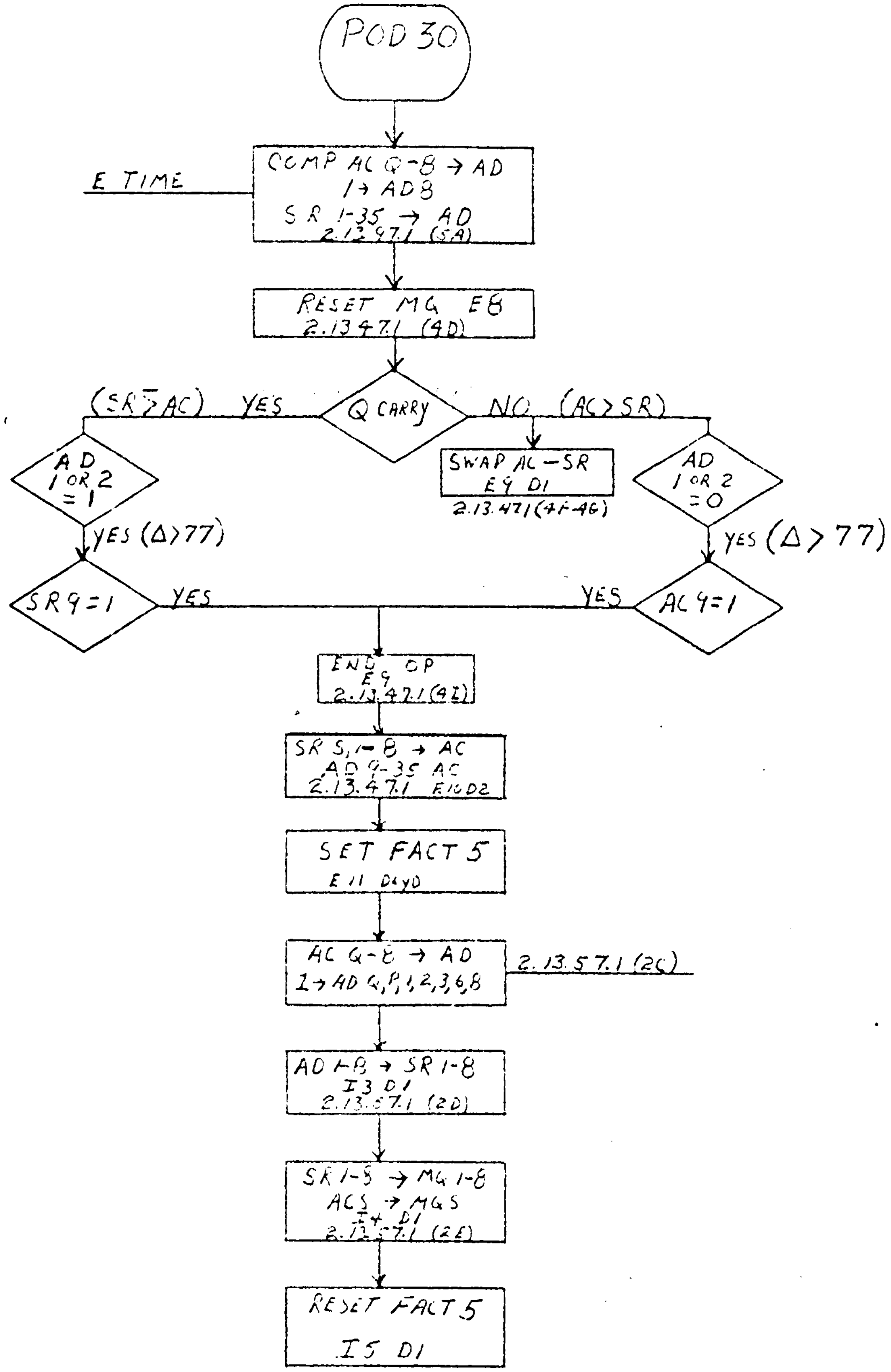
(SIGNS ALIKE – EQUAL CHARS. – AT LEAST ONE OPERAND NORMAL)



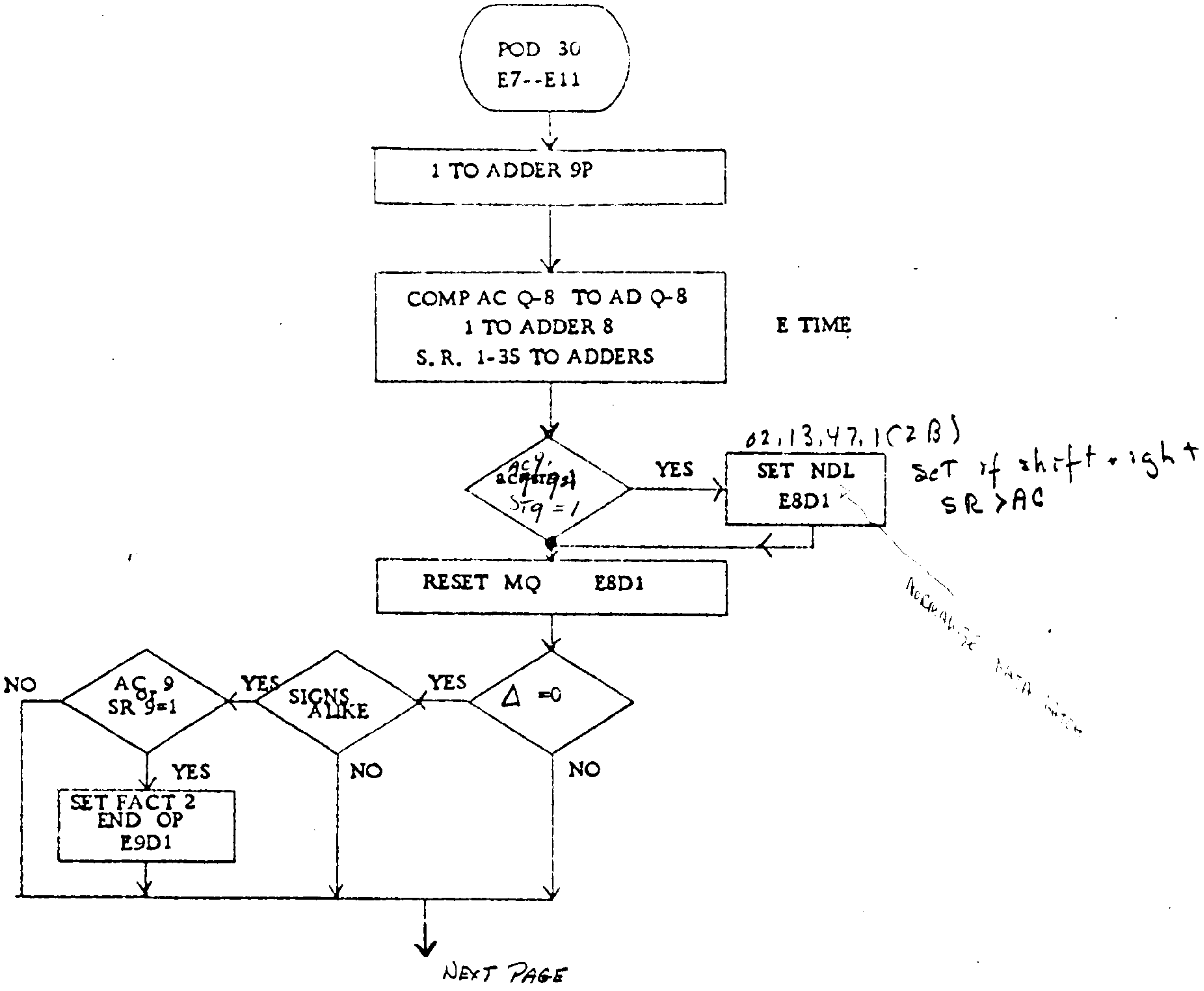
NOTE: GROUP 7 = 9P, 9, 10, 11

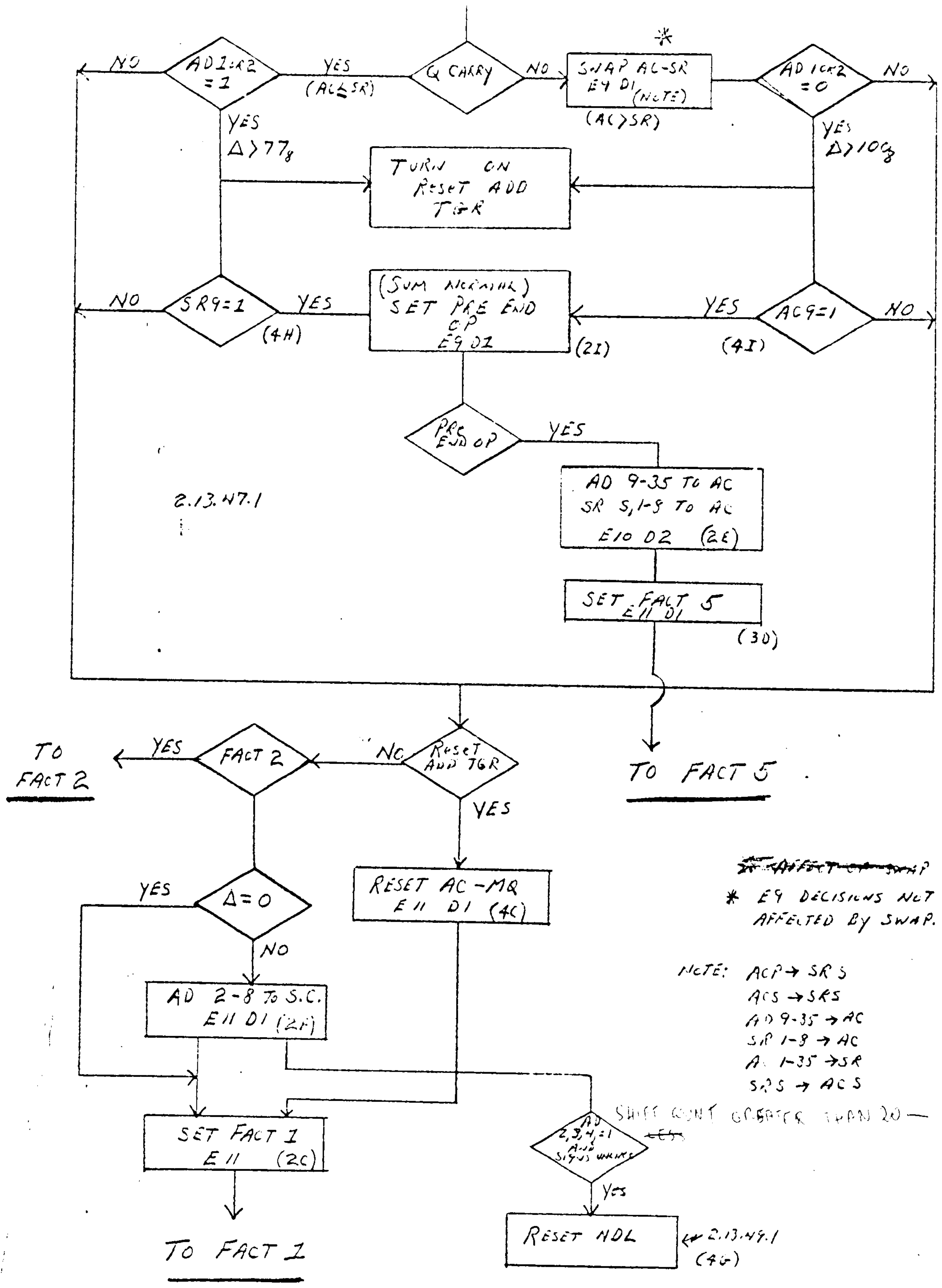
FAD - SECOND CASE

(BOTH STORAGE + AC NORMAL)
EXPONENT DIFFERENCE > 77_8



FLOATING ADD

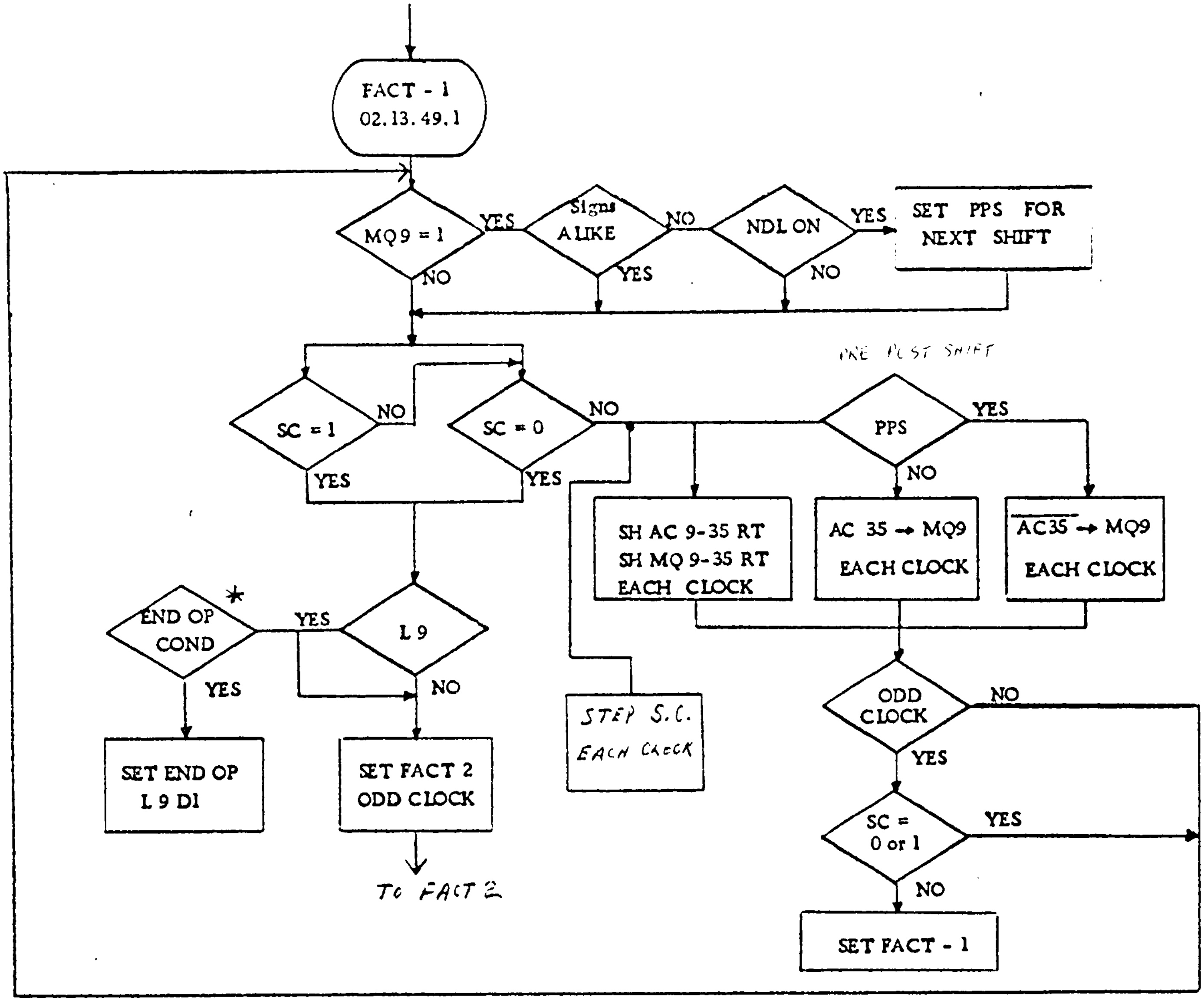




~~IF AFFECT OF SWAP~~

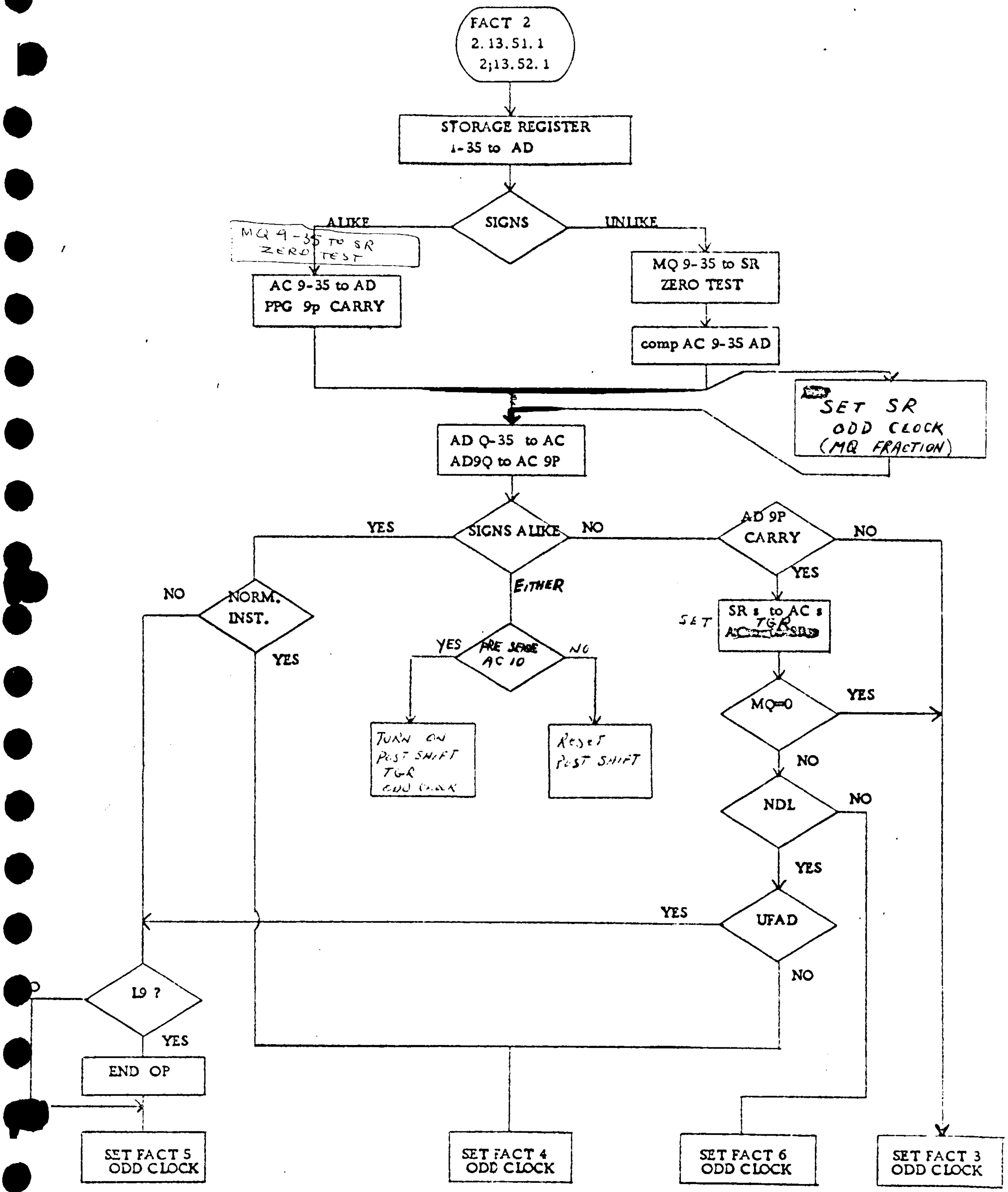
* E9 DECISIONS NOT AFFECTED BY SWAP.

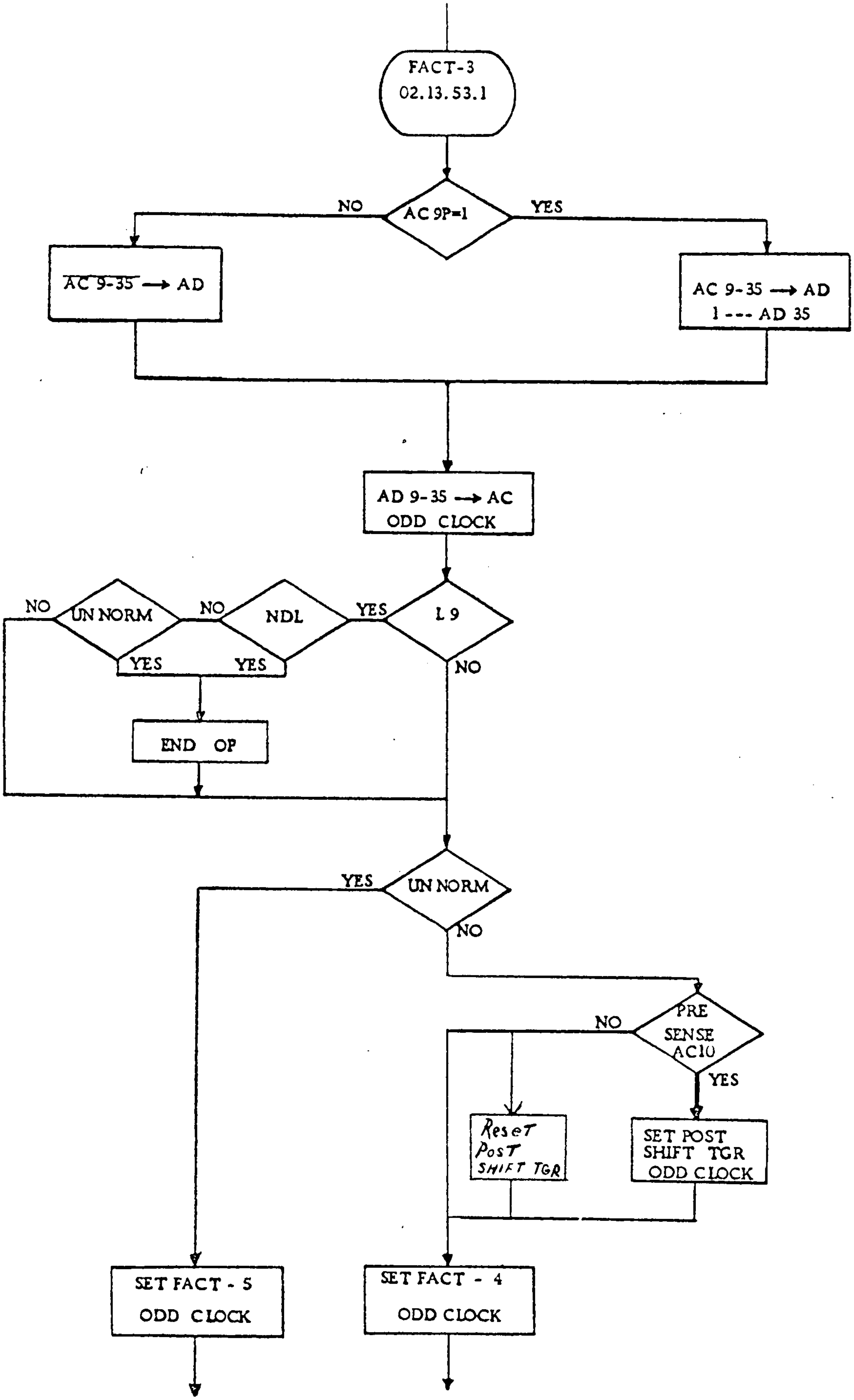
NOTE: ACP → SRS
ACS → SRS
AD 9-35 → AC
SR 1-8 → AC
AC 1-35 → SR
SRS → ACS

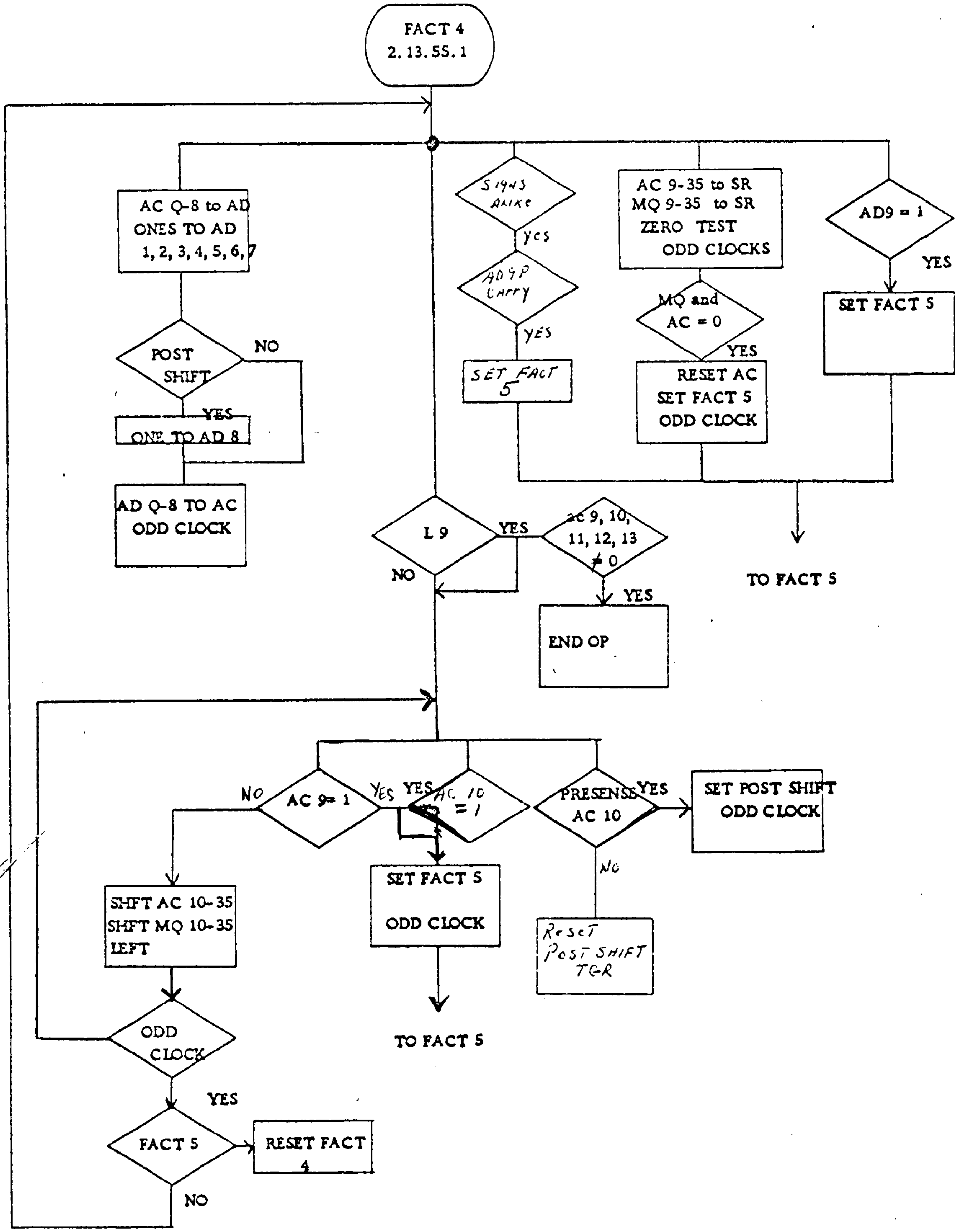


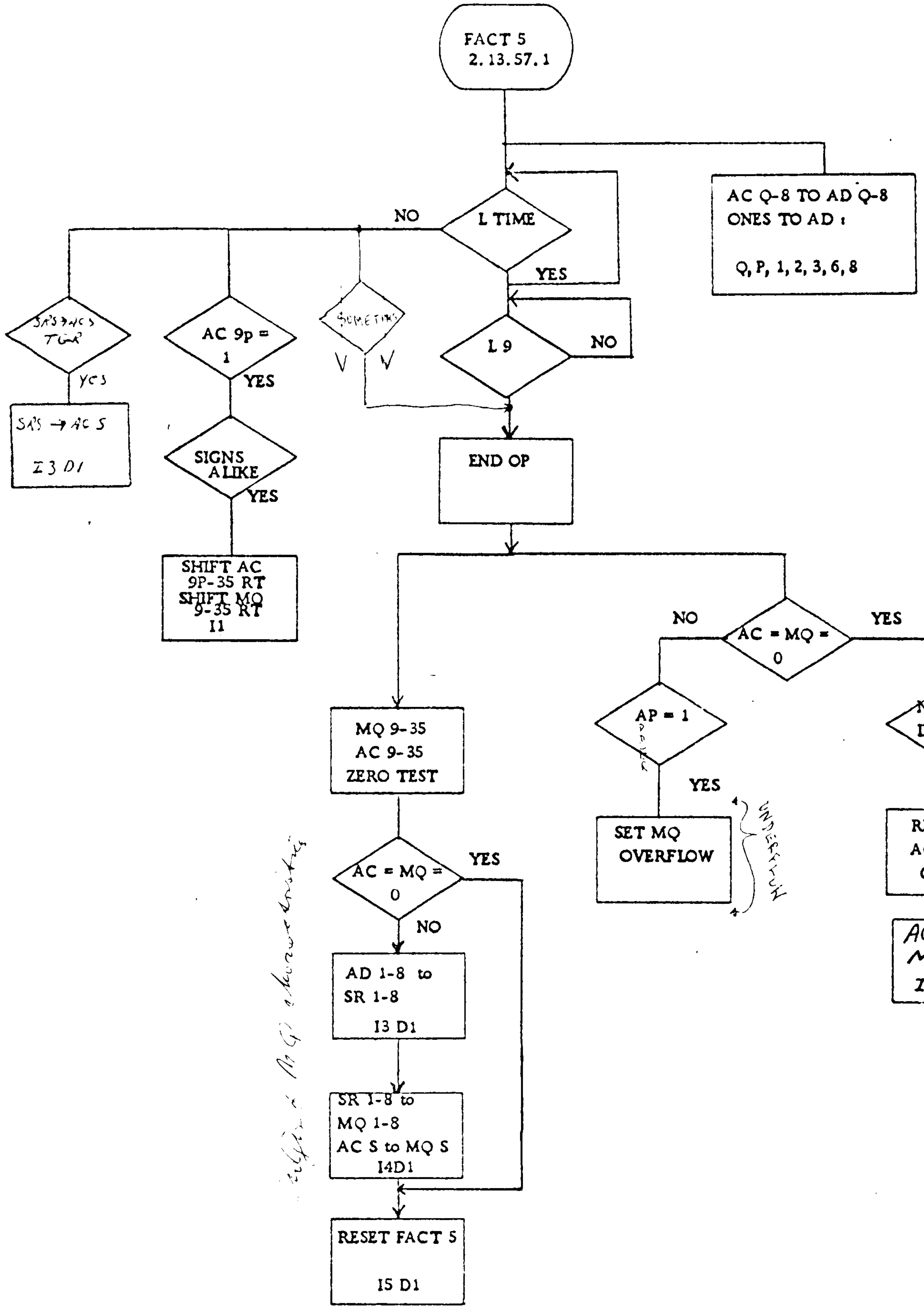
* END OP CONDITIONS

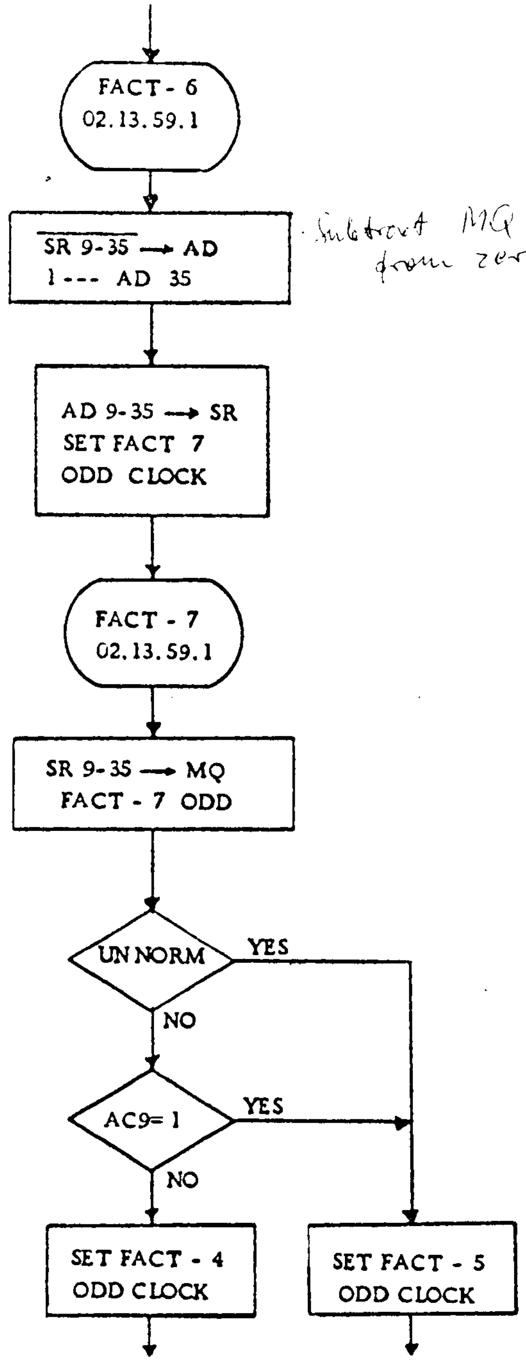
1. SIGNS ALIKE AND NDL ON
2. UNNORMALIZED INST. AND (SIGNS ALIKE OR NDL ON)


FACT 2
2. 13. 51. 1
2;13. 52. 1
STORAGE REGISTER
1-35 to AD
SIGNS
ALIKE
UNLIKE
MQ 9-35 TO SR
ZERO TEST
AC 9-35 to AD
PPG 9p CARRY
MQ 9-35 to SR
ZERO TEST
comp AC 9-35 AD
SET SR
ODD CLOCK
(MQ FRACTION)
AD Q-35 to AC
AD9Q to AC 9P
YES
SIGNS ALIKE
NO
AD 9P
CARRY
NO
YES
NO
NORM.
INST.
YES
EITHER
SET
SR s to AC s
TGR
YES
PRE SENSE
AC 10
NO
TURN ON
POST SHIFT
TGR
ODD CLOCK
RESET
POST SHIFT
MQ=0
YES
NO
NDL
NO
YES
YES
UFAD
NO
O
L9 ?
YES
END OP
SET FACT 5
ODD CLOCK
SET FACT 4
ODD CLOCK
SET FACT 6
ODD CLOCK
SET FACT 3
ODD CLOCK


FACT-3
02.13.53.1
AC 9P=1
NO
YES
$\overline{\text{AC 9-35}}$ → AD
AC 9-35 → AD
1 --- AD 35
AD 9-35 → AC
ODD CLOCK
NO
UN NORM
NO
NDL
YES
L 9
YES
YES
NO
END OP
YES
UN NORM
NO
NO
PRE SENSE AC10
YES
Reset Post SHIFT TGR
SET POST SHIFT TGR ODD CLOCK
SET FACT - 5
ODD CLOCK
SET FACT - 4
ODD CLOCK


FACT 4
2. 13. 55. 1
AC Q-8 to AD
ONES TO AD
1, 2, 3, 4, 5, 6, 7
POST
SHIFT
NO
YES
ONE TO AD 8
AD Q-8 TO AC
ODD CLOCK
Signs
Alike
yes
AD9P
Carry
yes
SET FACT
5
AC 9-35 to SR
MQ 9-35 to SR
ZERO TEST
ODD CLOCKS
MQ and
AC = 0
YES
RESET AC
SET FACT 5
ODD CLOCK
AD9 = 1
YES
SET FACT 5
TO FACT 5
L 9
YES
NO
ac 9, 10,
11, 12, 13
≠ 0
YES
END OP
NO
AC 9= 1
Yes
YES
AC 10
= 1
PRESENSE
AC 10
YES
SET POST SHIFT
ODD CLOCK
SHFT AC 10-35
SHFT MQ 10-35
LEFT
SET FACT 5
ODD CLOCK
No
Reset
Post SHIFT
TGR
TO FACT 5
ODD
CLOCK
YES
FACT 5
RESET FACT
4
NO

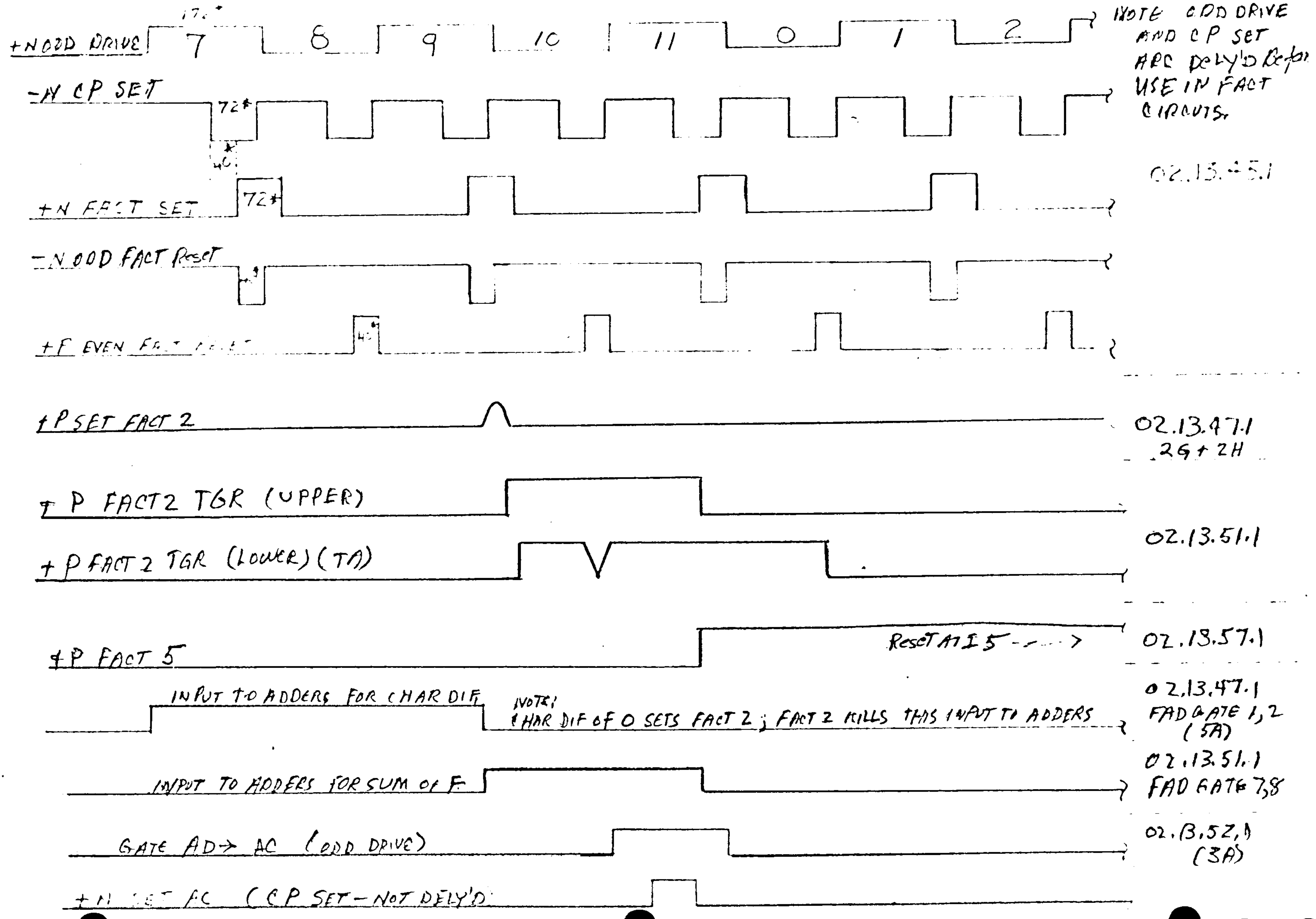

FACT 5
2.13.57.1
AC Q-8 TO AD Q-8
ONES TO AD :
Q, P, 1, 2, 3, 6, 8
L TIME
NO
YES
L 9
NO
SA'S ≠ AC S
TGR
YES
SA'S → AC S
I 3 D1
AC 9p =
1
YES
SIGNS
ALIKE
YES
SHIFT AC
9P-35 RT
SHIFT MQ
9-35 RT
I1
SOMETIMES
V
V
END OP
MQ 9-35
AC 9-35
ZERO TEST
AC = MQ =
0
YES
NO
AD 1-8 to
SR 1-8
I3 D1
SR 1-8 to
MQ 1-8
AC S to MQ S
I4D1
RESET FACT 5
I5 D1
NO
AC = MQ =
0
YES
AP = 1
YES
SET MQ
OVERFLOW
UNDERFLOW
NORM.
INST.
YES
RESET
AC & MQ
CHARS.
ACS →
MQ S
I 3 D1
MQ characteristic



Subtract MQ content from zero

SAMPLE TIMINGS – FAD

(SIGNS ALIKE – CHAR. EQUAL – AT LEAST ONE OPERAND NORMAL)



FAD 11/22/61 DS

FACT SEQUENCE	PRE-SHIFTS	POST-SHIFTS	CYCLES
IE-5	—	—	2
IE-1-2-5	9-10	—	3
IE-1-2-3-5	7-8	—	3
IE-1-2-3-4-5	4	~~6~~ 4	3
	3	~~6~~ 4	3
	2	~~8~~ 6	3
	1	~~8~~ 6	3
IE-1-2-4-5	6	~~6~~ 4	3
	5	~~6~~ 4	3
	4	~~8~~ 6	3
	3	~~8~~ 6	3
	2	~~10~~ 8	3
	1	~~10~~ 8	3
IE-1-2-6-7-5	1-2	—	3
IE 1-2-6-7-4-5	1-2	~~6~~ 4	3

FLOATING POINT MULTIPLY

In floating point multiply the contents of the storage register are multiplied by the contents of the multiplier quotient register. The product of the multiplication is placed in the AC and MQ. AC positions 1 - 8 contain the characteristic and positions 9 - 35 contain the 27 most significant bits of the fraction. MQ positions 1 - 8 contain the characteristic, less 27_{10}.

The 7094 computer simultaneously performs two distinct operations during a floating point multiply. instruction:

1. Addition of the characteristics.
2. Multiplication of the fractions.

The characteristics are added because they represent logarithmic notation. The fractions are multiplied as in a true binary multiplication. An example of this kind of multiplication would be the problem to multiply 5_{10} by 5_{10}, which would yield the answer of 25_{10}. Conversion to binary gives 101_2 x 101_2. In logarithmic or exponential notation the binary expression 101 becomes 011.101 (binary point to be moved three places to the right).

Adding the characteristics:

```
  011
 +011
 ----
  110
```

Multiplying the fractions:

```
    .101
  x .101
  ------
     101
    1010
 -------
 .011001
```

Adding the two products gives the floating point expression:

110.011001

The characteristic 110 indicates the binary point is to be moved six places to the right and the fraction .011001 becomes 011001. This is the correct answer because $011001_2 = 25_{10}$.

Handling the characteristic and fractions separately is exactly the way the 7094 performs this kind of problem. As previously stated, the action occurs simultaneously. The characteristic is computed during the first L cycle, at the same time part of the fractions are multiplied. Fraction multiplication is completed during subsequent L cycles and the two separate products added and the sign fixed during I time of the following instruction.

As in floating add, the value 200_8 is the dividing line between positive and negative numbers. Because both characteristics have this value, 200_8 is subtracted during the multiply sequence.

In the previous example it was assumed both numbers were unnormalized. Therefore, normalizing did not take place. Had both numbers been normalized, the product would have been normalized.

Following is a summary of computer activity during the execution of a floating point multiply (FMP) instruction:

1. Bring multiplicand into SR.

2. Check signs and set sign of AC to the algebraic sign of the product.

3. Add the characteristics of the MQ and SR.

4. Subtract 200_8 from the sum of the characteristics and place the remainder in the AC.

5. Multiply the SR fraction by the MQ fraction.

6. Set MQ characteristic equal to AC characteristic less 27_{10}.

Other steps taken by the computer but not covered in the summary include zero checks and normalizing. These will be shown on the flow chart of the instruction.

Figure 406.2-1 is a flow chart of the floating point multiply instruction sequence.

The following summary is based on the cycle of operation the computer is in during execution of the instruction.

E Time

The following steps are performed during E time:

1. Set SC to 33_8.

2. Bring the multiplicand from core storage and into the SR.

3. Bring up data gates between the SR and the AD. These will remain up during E time.

4. Perform a zero check on the MQ. Refer to the functional description of the storage register for the mechanics of this zero check.

5. Clear the AC and load AC positions 1 - 8 with the contents of SR positions 1 - 8.

6. Because SC 17 = 1, set SC odd trigger.

7. Check contents of MQ positions 34 and 35 and set appropriate trigger for first fraction multiplication.

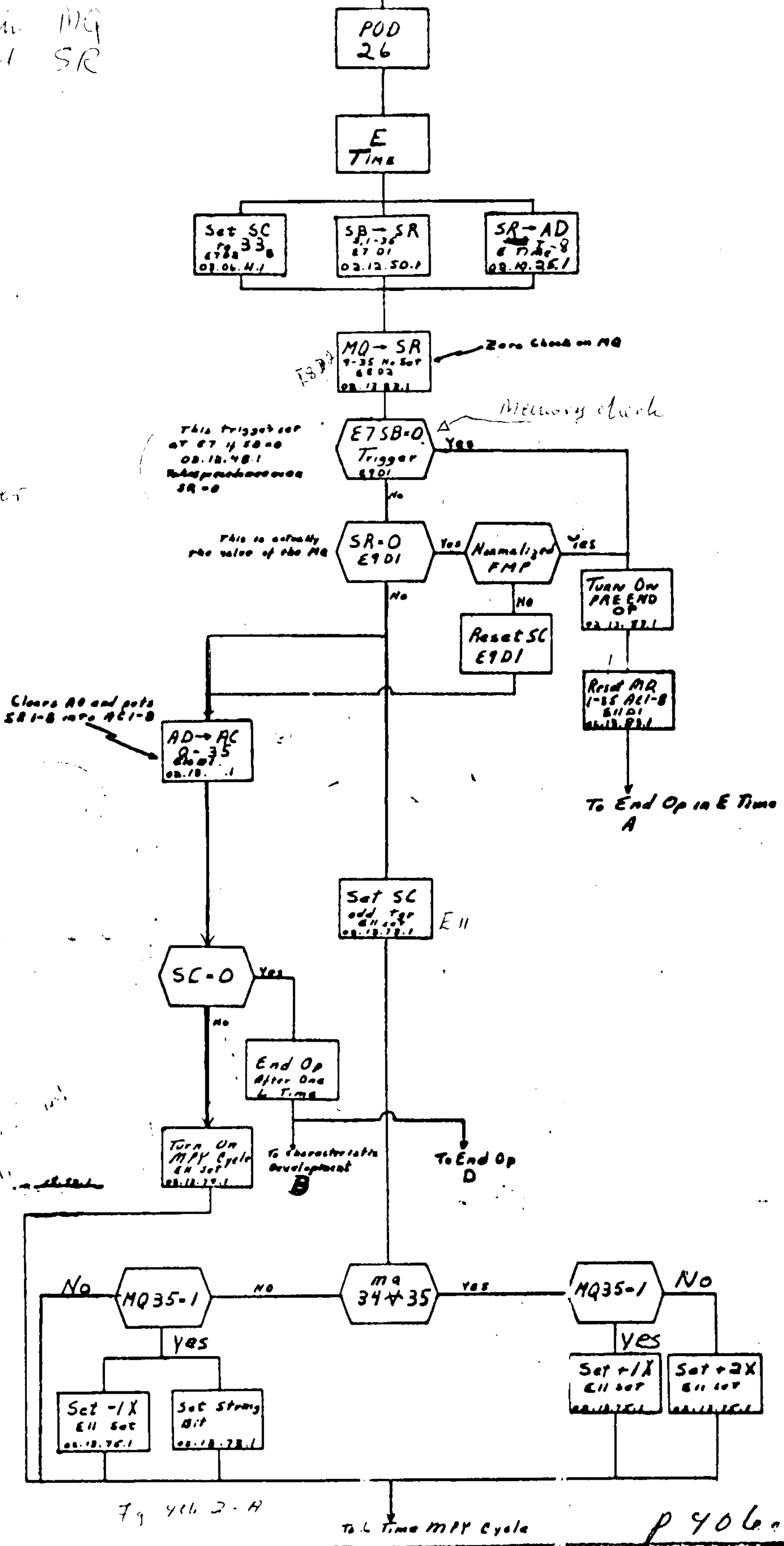
At the end of E time the computer is prepared for the first fraction multiplication. Multiplication takes place during subsequent L times, at a rate of 12 bits each cycle, i.e., 12 bits of the MQ are processed each L cycle. If the full 27 bits of the MQ are to be used, three L cycles will be required. This fixed the maximum number of cycles required for a single precision floating point multiplication at five -- one I, one E and three L cycles. The characteristic is processed during the first L cycle.

L Time

During the first L cycle the computer simultaneously processes the fraction and the characteristics. Notice on the flow chart that the tally counter count must be one to enable the computer to add the characteristics. TC is one during the first L time of FMP. The tally counter will be stepped at the end of the first L cycle. When TC = 2, the computer is blocked from further processing of the characteristic.

Multiplier in MQ
Multiplicand SR

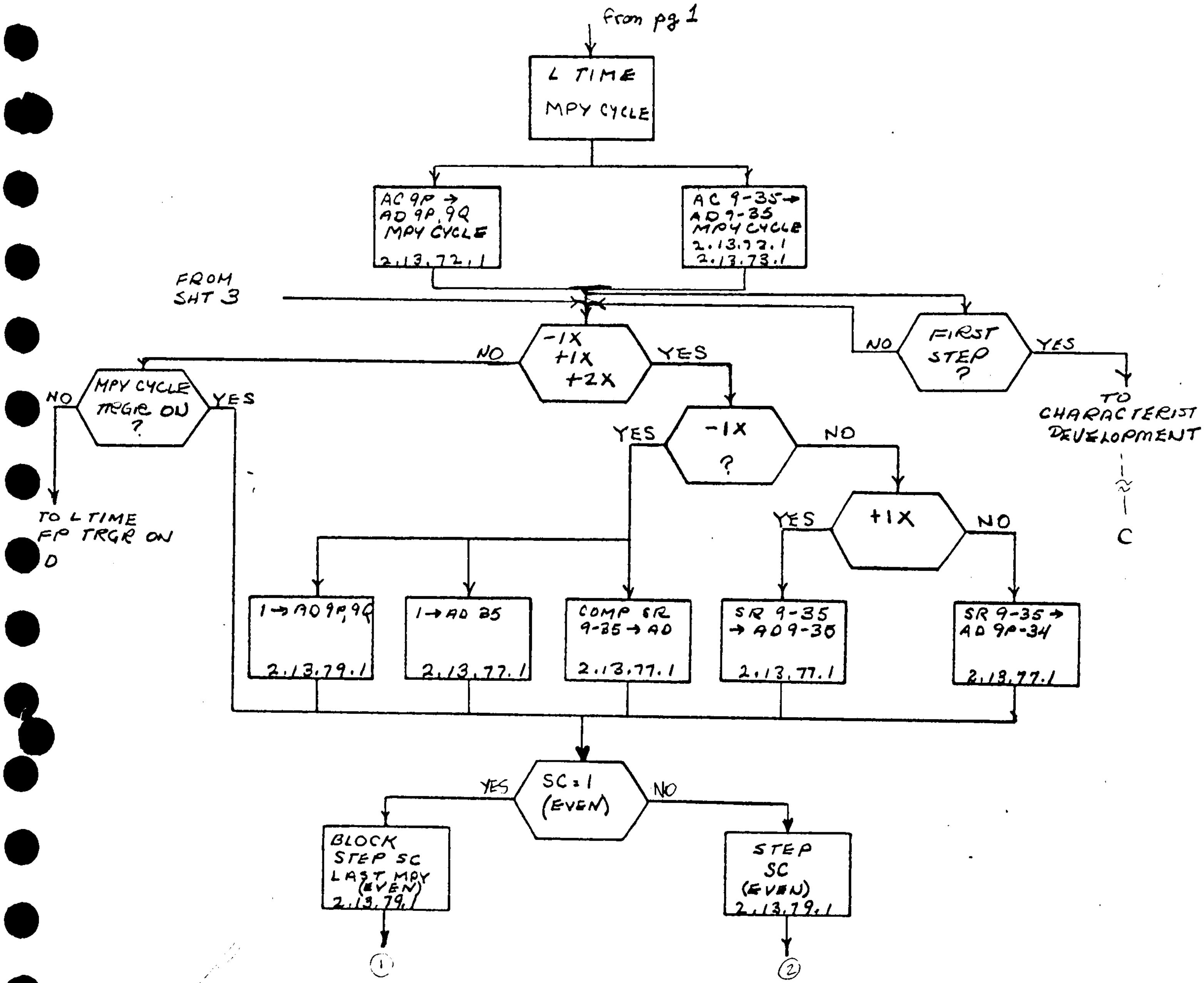
this trigger set
at E7 if SB = 0
02.12.48.1
Takes precedence over
SR = 0



POD 26
E Time
Set SC to 33₈ E7D2 02.06.41.1
SB → SR 1-35 E7 D1 02.12.50.1
SR → AD 1-8 E Time 02.10.35.1
MQ → SR 9-35 No Set E8D2 02.12.83.1
E8D2
Zero Check on MQ
E7 SB = 0 Trigger E9D1
Memory check
This trigger set at E7 if SB = 0 02.12.48.1 Takes precedence over SR = 0
Yes
No
SR = 0 E9D1
This is actually the value of the MQ
Yes
Normalized FMP
Yes
No
No
Reset SC E9D1
Turn On PRE END OP 02.12.83.1
Reset MQ 1-35 AC1-8 E11D1 02.12.83.1
To End Op in E Time A
Clears AC and puts SR 1-8 into AC1-8
AD → AC 0-35 Reset 02.12. .1
Set SC odd tgr E11 set 02.12.73.1
E 11
SC = 0
Yes
No
End Op After One L Time
Turn On MPY Cycle E11 set 02.12.79.1
To Characteristic Development B
To End Op D
MQ 34 ∀ 35
NO
Yes
MQ35 = 1
No
yes
MQ35 = 1
No
yes
Set -1X E11 Set 02.12.76.1
Set String Bit 02.12.73.1
Set +1X E11 set 02.12.76.1
Set +2X E11 set 02.12.76.1
To L Time MPY Cycle

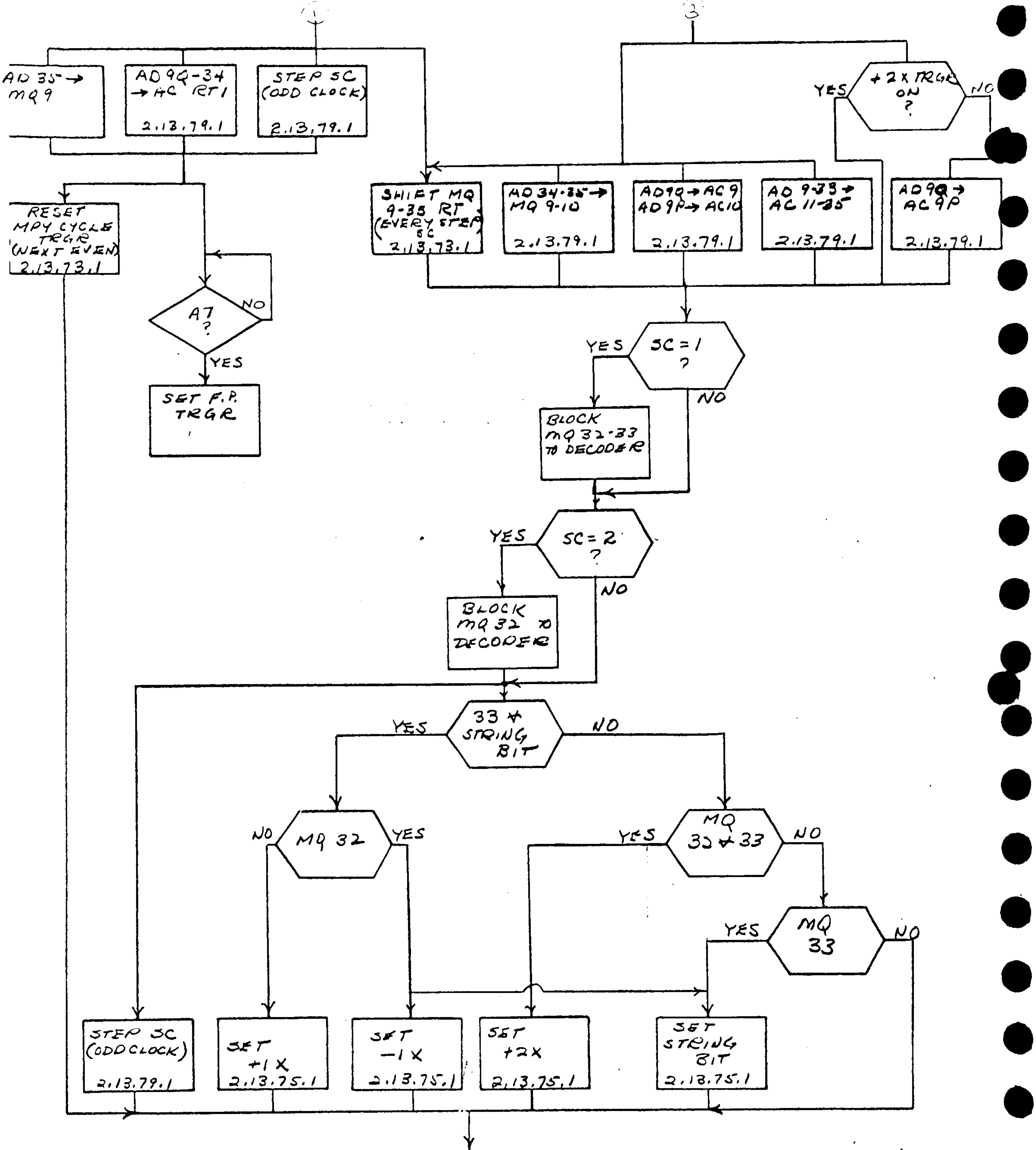
Fig 406 2-A

p 406.35


from pg 1
L TIME MPY CYCLE
AC 9P → AD 9P, 9Q MPY CYCLE 2.13.72.1
AC 9-35 → AD 9-35 MPY CYCLE 2.13.72.1 2.13.73.1
FROM SHT 3
NO
-1X +1X +2X
YES
NO
FIRST STEP ?
YES
TO CHARACTERIST DEVELOPMENT
C
NO
MPY CYCLE TRGR ON ?
YES
TO L TIME FP TRGR ON
D
YES
-1X ?
NO
YES
+1X
NO
1 → AD 9P, 9Q 2.13.79.1
1 → AD 35 2.13.77.1
COMP SR 9-35 → AD 2.13.77.1
SR 9-35 → AD 9-36 2.13.77.1
SR 9-35 → AD 9P-34 2.13.77.1
YES
SC=1 (EVEN)
NO
BLOCK STEP SC LAST MPY (EVEN) 2.13.79.1
STEP SC (EVEN) 2.13.79.1
1
2

To Sheet 3

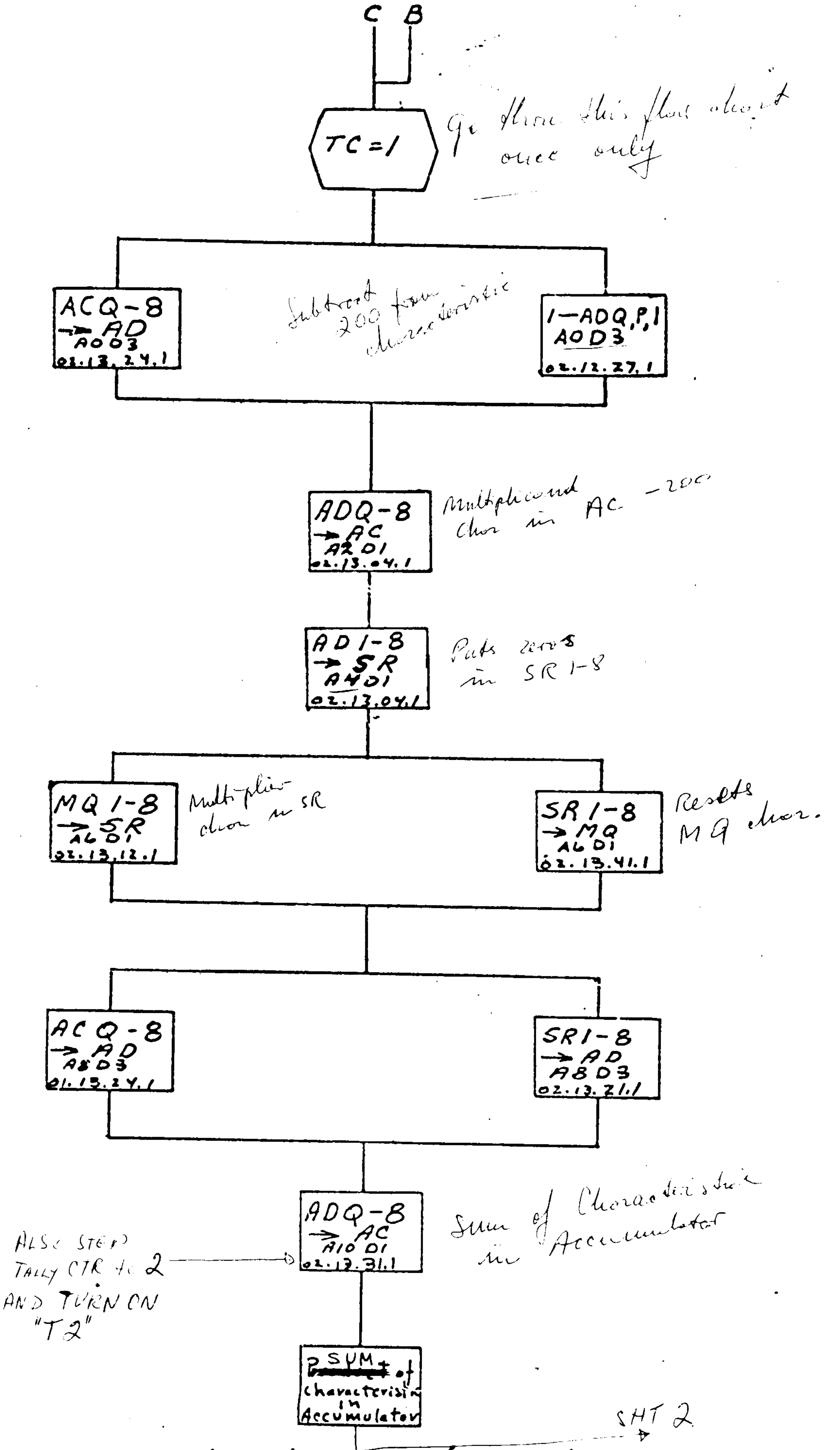
SINGLE PRECISION
FLOATNG MULTIPLY
SHT #2



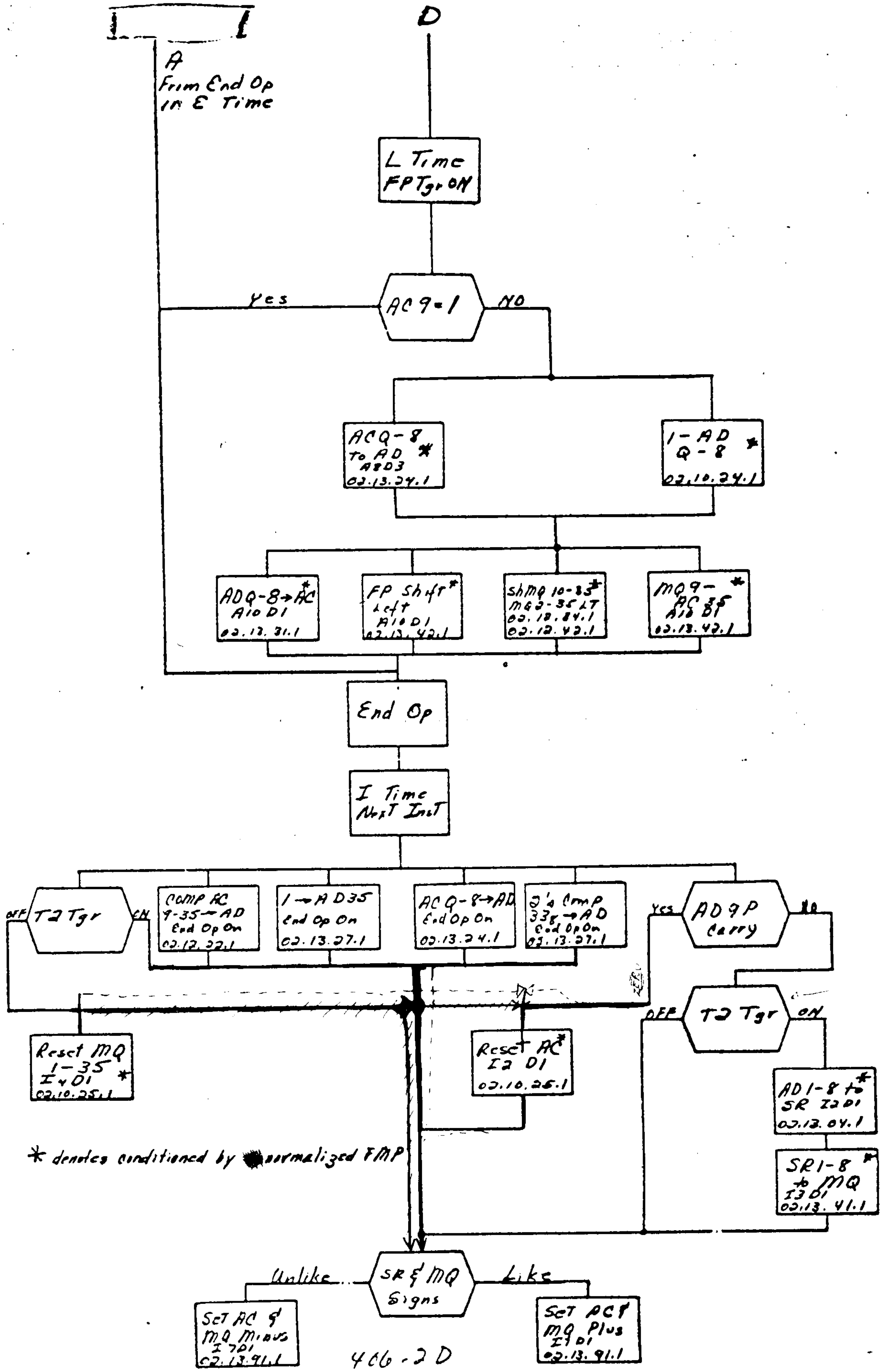
TO SHEET 2
+1X, −1X, +2X

SINGLE PRECISION
FLOATING MULTIPLY
SHT # 3

406,36A


C
B
TC=1
Go thru this flow chart once only
ACQ-8 → AD A0D3 02.13.24.1
Subtract 200 from characteristic
1—ADQ,P,1 A0D3 02.12.27.1
ADQ-8 → AC A2D1 02.13.04.1
Multiplicand char in AC -200
AD1-8 → SR A4D1 02.13.04.1
Puts zeros in SR 1-8
MQ1-8 → SR A6D1 02.13.12.1
Multiplier char in SR
SR1-8 → MQ A6D1 02.13.41.1
Resets MQ char.
ACQ-8 → AD A8D3 01.13.24.1
SR1-8 → AD A8D3 02.13.21.1
ADQ-8 → AC A10D1 02.13.31.1
Sum of Characteristic in Accumulator
ALSO STEP TALLY CTR to 2 AND TURN ON "T2"
~~Product~~ SUM of characterisic in Accumulator
SHT 2

Characteristic Development — FMP ~~406 2C~~ 406.3



* denotes conditioned by normalized FMP

406-2D

If on means
* no L time

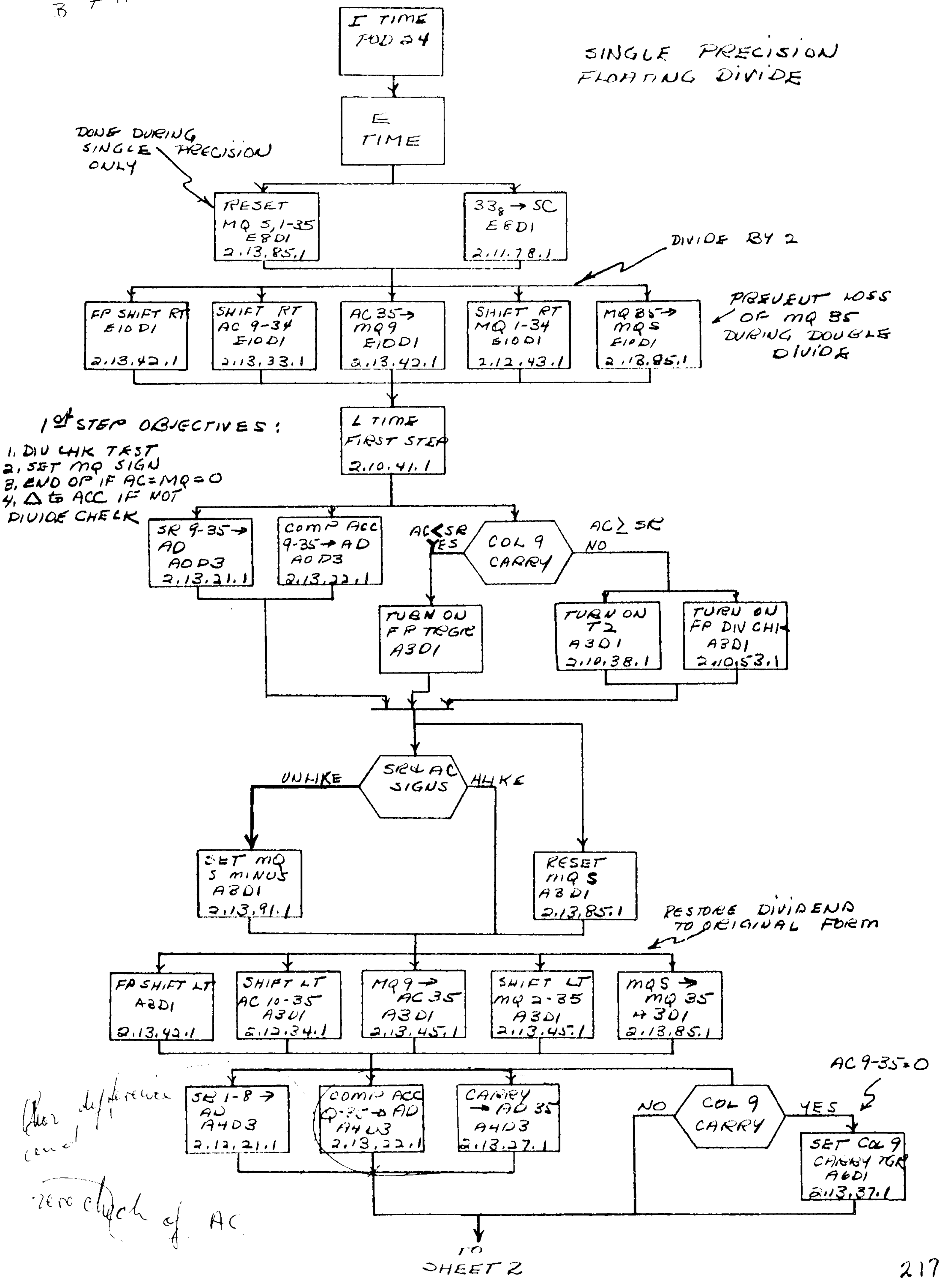
p 406.38

A · B = C

$\frac{C}{B} \neq A$

PAGE 1

SINGLE PRECISION FLOATING DIVIDE



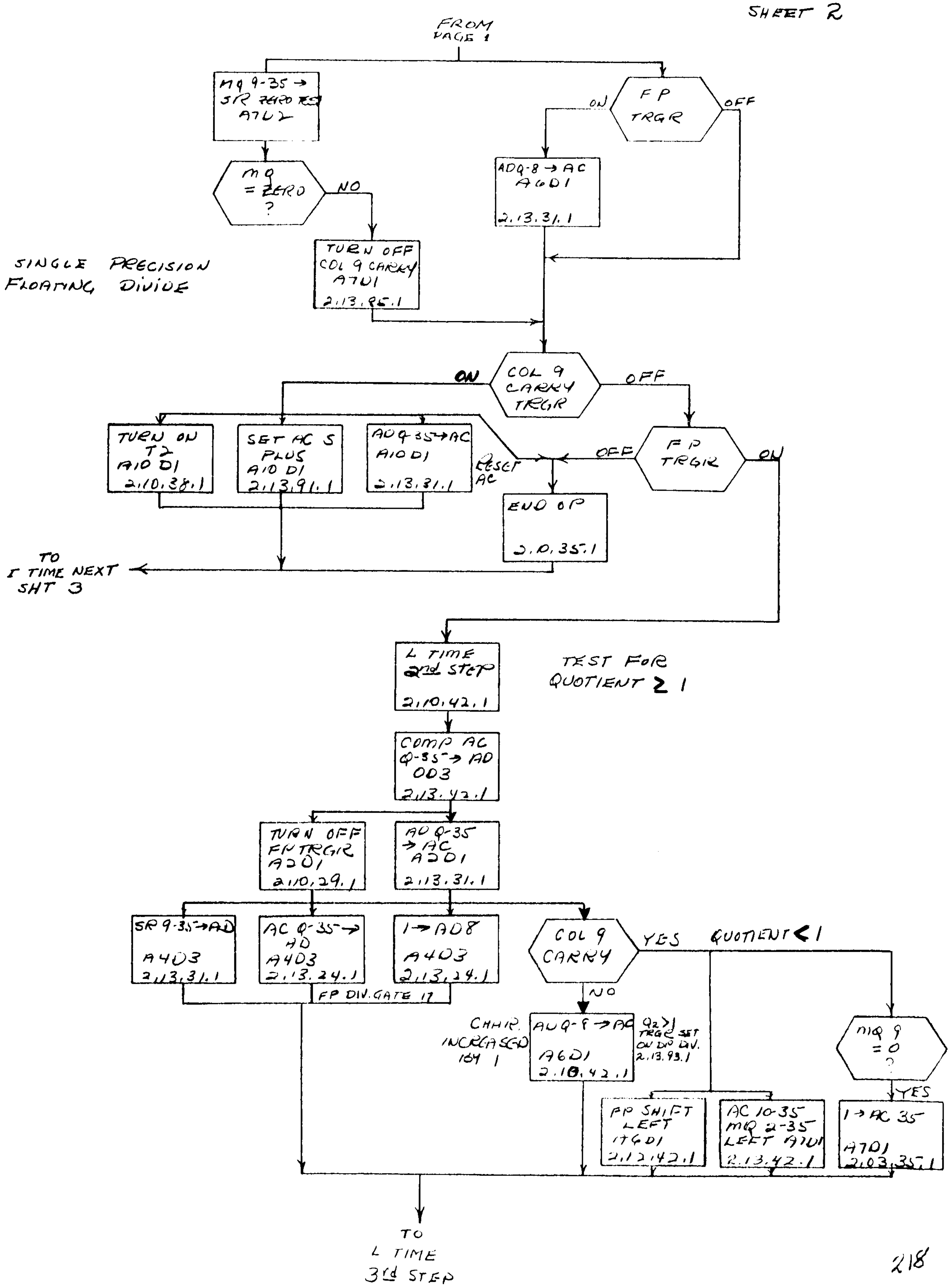
I TIME POD 24
E TIME
DONE DURING SINGLE PRECISION ONLY
RESET MQ S,1-35 E8D1 2.13.85.1
33_8 → SC E8D1 2.11.78.1
DIVIDE BY 2
FP SHIFT RT E10D1 2.13.42.1
SHIFT RT AC 9-34 E10D1 2.13.33.1
AC 35 → MQ 9 E10D1 2.13.42.1
SHIFT RT MQ 1-34 E10D1 2.12.43.1
MQ 35 → MQ S E10D1 2.13.85.1
PREVENT LOSS OF MQ 35 DURING DOUBLE DIVIDE
1st STEP OBJECTIVES:
1. DIV CHK TEST
2. SET MQ SIGN
3. END OP IF AC=MQ=0
4. Δ to ACC IF NOT DIVIDE CHECK
L TIME FIRST STEP 2.10.41.1
SR 9-35 → AD AOD3 2.13.21.1
COMP ACC 9-35 → AD AOD3 2.13.22.1
AC<SR YES
COL 9 CARRY
AC≥SR NO
TURN ON FP TRGR A3D1
TURN ON T2 A3D1 2.10.38.1
TURN ON FP DIV CHK A3D1 2.10.53.1
UNLIKE
SR & AC SIGNS
ALIKE
SET MQ S MINUS A8D1 2.13.91.1
RESET MQ S A8D1 2.13.85.1
RESTORE DIVIDEND TO ORIGINAL FORM
FP SHIFT LT A3D1 2.13.42.1
SHIFT LT AC 10-35 A3D1 2.12.34.1
MQ 9 → AC 35 A3D1 2.13.45.1
SHIFT LT MQ 2-35 A3D1 2.13.45.1
MQ S → MQ 35 A3D1 2.13.85.1
AC 9-35=0
SR 1-8 → AD A4D3 2.13.21.1
COMP ACC 9-35 → AD A4D3 2.13.22.1
CARRY → AD 35 A4D3 2.13.27.1
NO
COL 9 CARRY
YES
SET COL 9 CARRY TGR A6D1 2.13.37.1
TO SHEET 2

Char difference and

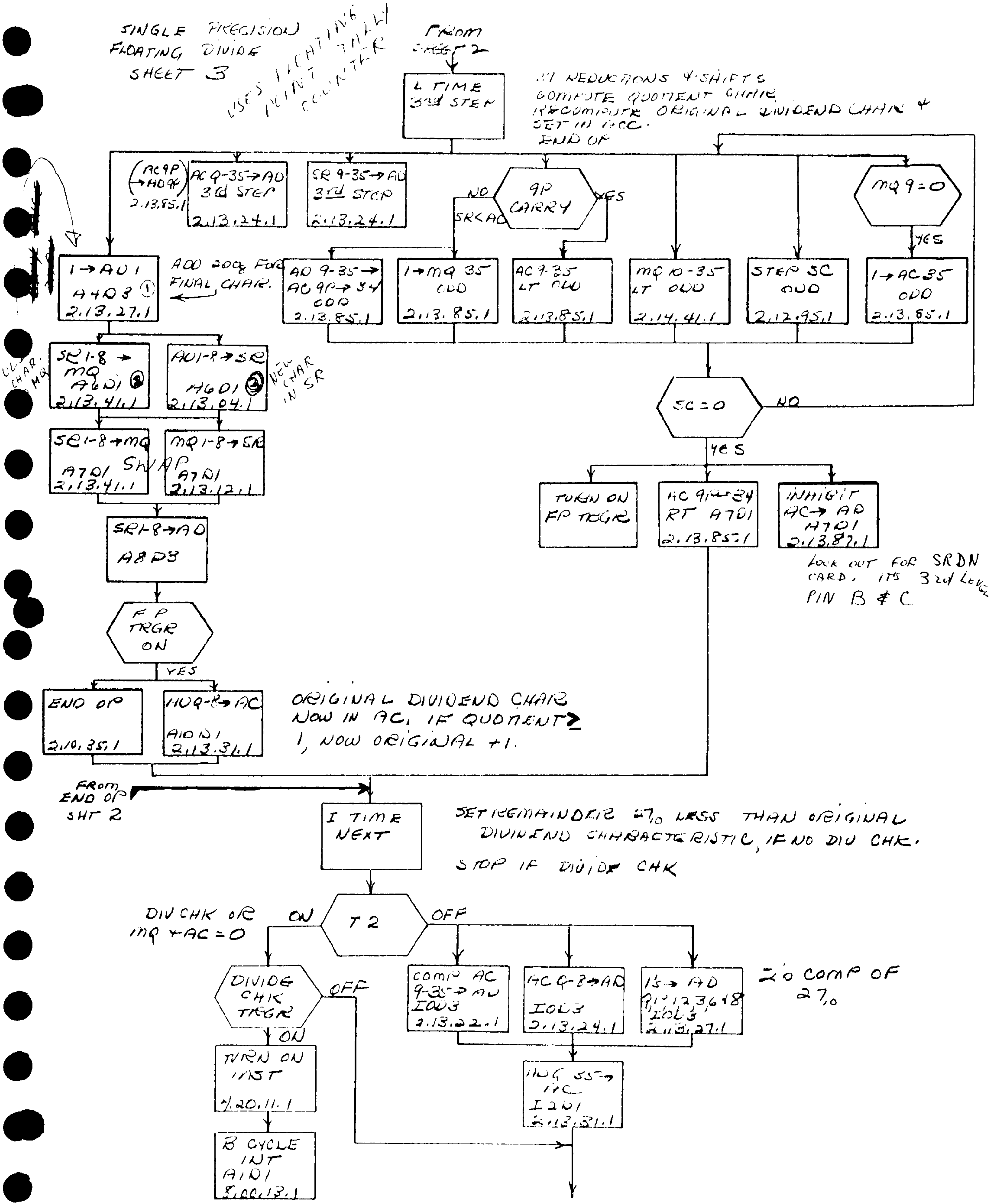
zero check of AC

SHEET 2

SINGLE PRECISION FLOATING DIVIDE


FROM PAGE 1
MQ 9-35 → SR ZERO TEST A7D2
MQ = ZERO ?
NO
TURN OFF COL 9 CARRY A7D1 2.13.85.1
FP TRGR
ON
OFF
ADQ-8 → AC A6D1 2.13.31.1
COL 9 CARRY TRGR
ON
OFF
TURN ON T2 A10 D1 2.10.38.1
SET AC S PLUS A10 D1 2.13.91.1
ADQ-35 → AC A10 D1 2.13.31.1
RESET AC
FP TRGR
OFF
ON
END OP 2.10.35.1
TO I TIME NEXT SHT 3
L TIME 2nd STEP 2.10.42.1
TEST FOR QUOTIENT ≥ 1
COMP AC Q-35 → AD OD3 2.13.42.1
TURN OFF FP TRGR A2D1 2.10.29.1
AD Q-35 → AC A2D1 2.13.31.1
SR 9-35 → AD A4D3 2.13.31.1
AC Q-35 → AD A4D3 2.13.24.1
1 → AD8 A4D3 2.13.24.1
FP DIV. GATE 17
COL 9 CARRY
YES
QUOTIENT < 1
NO
CHAR. INCREASED BY 1
ADQ-8 → AC A6D1 2.10.42.1
Q2 > 1 TRGR SET ON DP DIV. 2.13.93.1
MQ 9 = 0 ?
YES
FP SHIFT LEFT 1TGD1 2.12.42.1
AC 10-35 MQ 2-35 LEFT A7D1 2.13.42.1
1 → AC 35 A7D1 2.03.35.1
TO L TIME 3rd STEP

SINGLE PRECISION FLOATING DIVIDE SHEET 3


USES FLOATING POINT TALLY COUNTER
FROM SHEET 2
L TIME 3rd STEP
27 REDUCTIONS & SHIFTS COMPUTE QUOTIENT CHAR & RECOMPUTE ORIGINAL DIVIDEND CHAR & SET IN ACC. END OF
(AC 9P → AD 9P) 2.13.85.1
AC 9-35 → AD 3rd STEP 2.13.24.1
SR 9-35 → AD 3rd STEP 2.13.24.1
9P CARRY
NO
SR<AC
YES
MQ 9=0
YES
1 → AU1 A4D3 ① 2.13.27.1
ADD 200₈ FOR FINAL CHAR.
AD 9-35 → AC 9P → 34 ODD 2.13.85.1
1 → MQ 35 ODD 2.13.85.1
AC 9-35 LT ODD 2.13.85.1
MQ 10-35 LT ODD 2.14.41.1
STEP SC ODD 2.12.95.1
1 → AC 35 ODD 2.13.85.1
OLD CHAR. TO MQ
SR 1-8 → MQ A6D1 ② 2.13.41.1
AU1-8 → SR A6D1 ② 2.13.04.1
NEW CHAR IN SR
SC=0
NO
YES
SR 1-8 → MQ A7D1 2.13.41.1
MQ 1-8 → SR A7D1 2.13.12.1
SWAP
TURN ON FP TRGR
AC 9P → 34 RT A7D1 2.13.85.1
INHIBIT AC → AD A7D1 2.13.87.1
LOOK OUT FOR SRDN CARD, ITS 3rd LEVEL PIN B & C
SR 1-8 → AD A8D3
FP TRGR ON
YES
END OP 2.10.35.1
MQ-8 → AC A10D1 2.13.31.1
ORIGINAL DIVIDEND CHAR NOW IN AC, IF QUOTIENT ≥ 1, NOW ORIGINAL +1.
FROM END OP SHT 2
I TIME NEXT
SET REMAINDER 27₀ LESS THAN ORIGINAL DIVIDEND CHARACTERISTIC, IF NO DIV CHK.
STOP IF DIVIDE CHK
DIV CHK OR MQ + AC = 0
ON
T 2
OFF
DIVIDE CHK TRGR
OFF
ON
COMP AC 9-35 → AD I0D3 2.13.22.1
AC 9-8 → AD I0D3 2.13.24.1
1's → AD 9P 1 2 3 6 + 8 I0D3 2.13.27.1
2's COMP OF 27₀
TURN ON INST 4.20.11.1
MQ 35 → AC I2D1 2.13.31.1
B CYCLE INT A1D1 8.00.13.1

This instruction adds $A \cdot 2^n + B \cdot 2^{n-27}$ to $C \cdot 2^m + D \cdot 2^{m-27}$ and places their sum in the ACC and MQ with the resultant characteristic in the ACC and a characteristic smaller by 27_{10} in the MQ. It assumes that $A \cdot 2^n$ and $B \cdot 2^{n-27}$ have been previously placed in the ACC and MQ respectively and that $C \cdot 2^m$ and $D \cdot 2^{m-27}$ are in consecutive even and odd locations in memory.

There are two sets of controls used to implement this instruction. First are the double precision synchronizer controls (DPS), three position step ring used to control data movement before and after additions. The second set of controls are the single precision floating add control triggers (FACT) that are used with some modifications. The following is a general description of these controls.

DPS0

DPS 0 controls the E time and first L0 and L1 time movements of data in preparation for the first add. At the end of L1 time the larger of the two operands will be in the SI and SR, the smaller in the MQ and ACC so they will appear in either of these configurations.

	SI	SR	ACC	MQ
1	A	$B \cdot 2^n$	$D \cdot 2^m$	C
2	C	$D \cdot 2^m$	$B \cdot 2^n$	A

An E time end operation may occur if the characteristic difference is greater than 77_8 and the larger of the two operands is normalized. If this occurs the larger of the two operands is placed in the ACC and MQ and FACT 5 is set to complete normal I time operations. If the characteristic difference is not excessive as described above FACT 1 is set for pre-normalization or FACT 2 for addition. All E time single precision operations are blocked except SR to adder gating, E9D1, ACC and SR exchange, AD to SC gate and the operation mentioned above. The DPS is stepped at L1 time.

DPS1

DPS1 controls pre-normalization (FACT 1) and first add (FACT 2). The special end operations of FACT 1 and 2 and the FACT sets of FACT 2 are blocked. During FACT 1 MQ 35 is shifted to ACC 9 and the ACC 35 to MQ9 gates are blocked. During FACT 2 a two's complement add is performed if the signs are unlike instead of the one's complement add of single precision. Single precision controls place the contents of MQ 9-35 in the SR during the odd clock pulse of FACT-2. The AC and SR fractions are swapped during the even and odd clock pulses following FACT-2 and the DPS is stepped to complete the operand relocation in preparation for the second addition.

p 4c8, 0.1

DPS2

DPS 2 is used to complete the operand relocation in preparation for the second add. On the odd clock pulse SI 9-35 and SR 9-35 are exchanged, the DPS is stepped and FACT 2 is set for the second addition.

DPS3

DPS 3 controls the second add, MQ adjust, normalization and end operation functions. During FACT 2 any 9 carries that were generated under DPS 2 control are now treated as carries into AD 35. If a true add is being performed the operation will proceed to FACT 4 for normalization or FACT 5 if normalization is not required. All single precision decisions during these controls are valid. If a complement add is being performed in FACT 2 the single precision decisions do not apply as they are based on a one's complement addition. If a 9 carry occurs during this complement add it indicates that the answer in both the MQ and ACC are in true form except that their contents must be checked for zero to determine the sign. If this condition occurs FACT 4 is set for post normalization. If a complement add is being performed and a 9 carry does not occur it indicates that the MQ and ACC are in two's complement form and must be corrected, so MQ 9-35 is set into SR 9-35 by the single precision controls of FACT 2 and FACT 6 is set. During FACT 6 the complement of SR 9-35 is gated to AD 9-35 along with a carry to AD 35 and on the odd clock pulse AD 9-35 is gated to SR 9-35. The carry save trigger is turned on if a carry resulted and FACT 7 is set. During FACT 7 SR 9-35 is gated to MQ 9-35 and ACC 9-35 is gated to AD 9-35 along with the carry save trigger. On the odd clock pulse AD 9-35 is gated to AC 9-35 and FACT 4 or 5 is set depending on normalization requirements. The instruction is then completed under single precision controls.

~~D. E. SPENCER~~
~~December 7, 1961~~

DFAD

PROBLEM: ADD

$$\begin{array}{r} A^{N} + B^{N-27} \\ + \; C^{M} + D^{M-27} \\ \hline \end{array}$$

FIRST STEP: ADD

$$\begin{array}{r} B \\ + \; D \\ \hline \end{array}$$

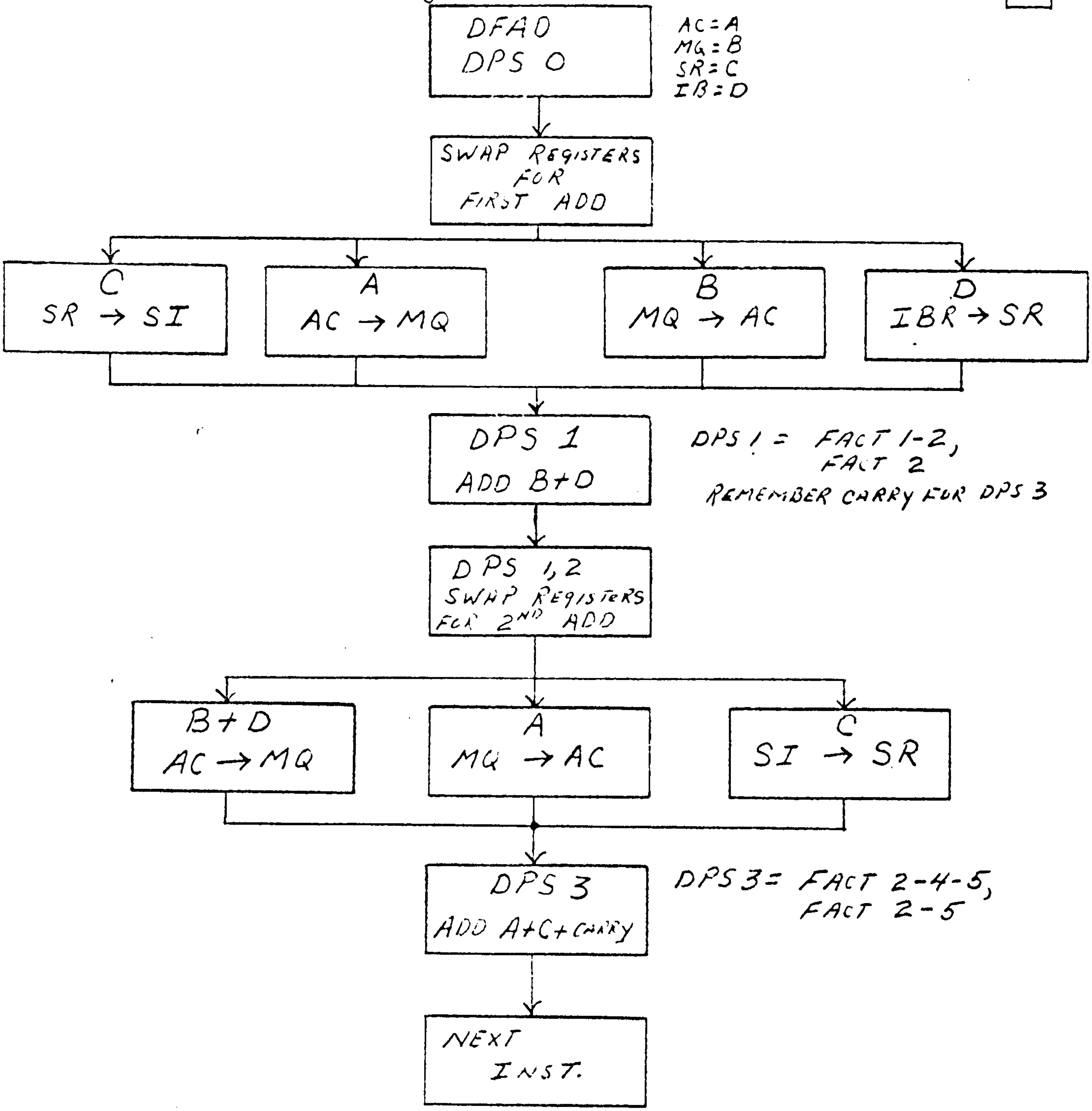
REMEMBER CARRY.
PLACE SUM OF B+D IN MQ

SECOND STEP: ADD

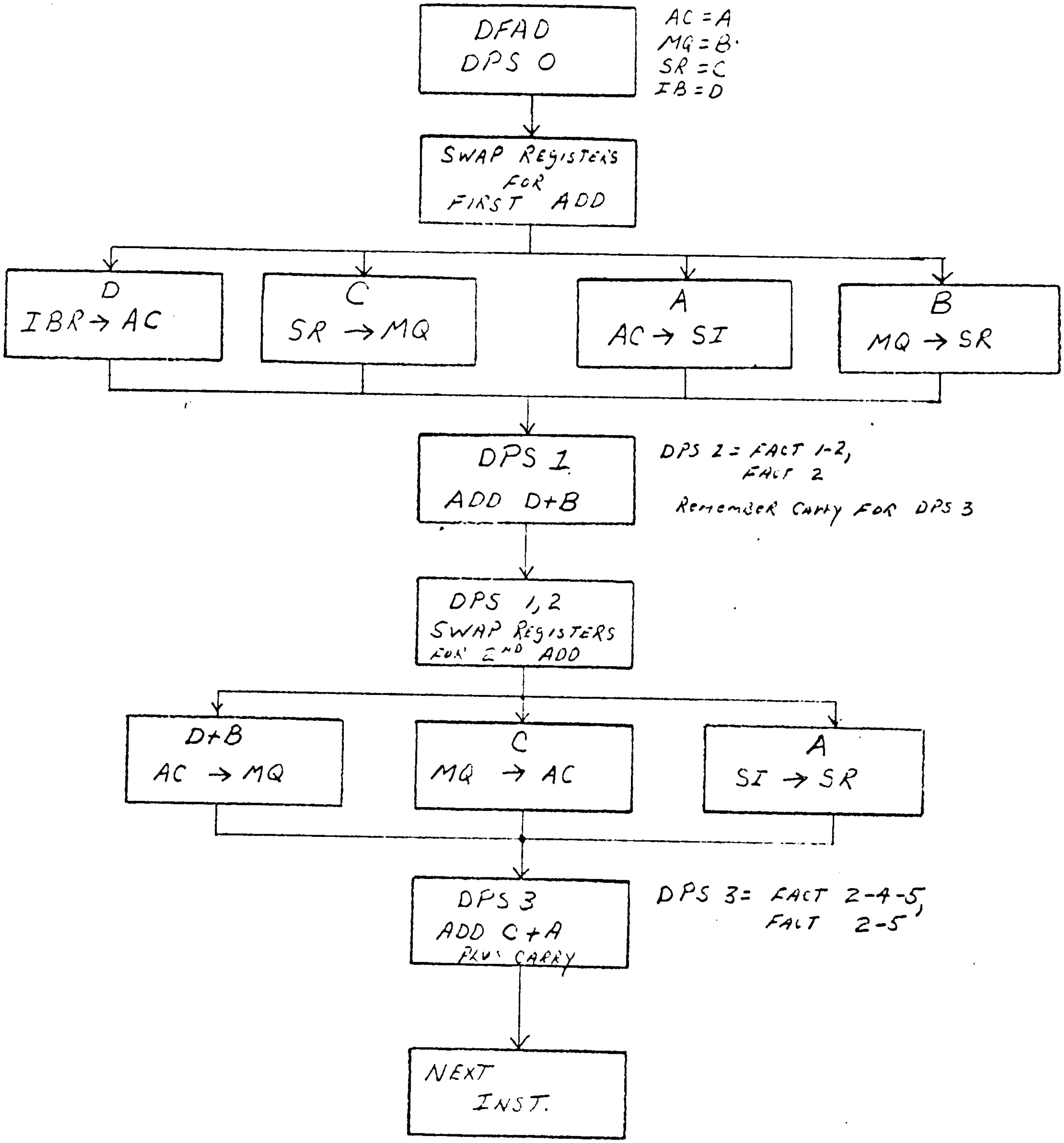
$$\begin{array}{r} A \\ + \; C \\ \hline \end{array}$$

ADD IN THE REMEMBERED CARRY.
PLACE SUM OF A+C IN AC.

DFAD - SIMPLIFIED DATA FLOW
SIGNS ALIKE SR > AC



DFAD - SIMPLIFIED DATA FLOW
SIGNS ALIKE SR< AC

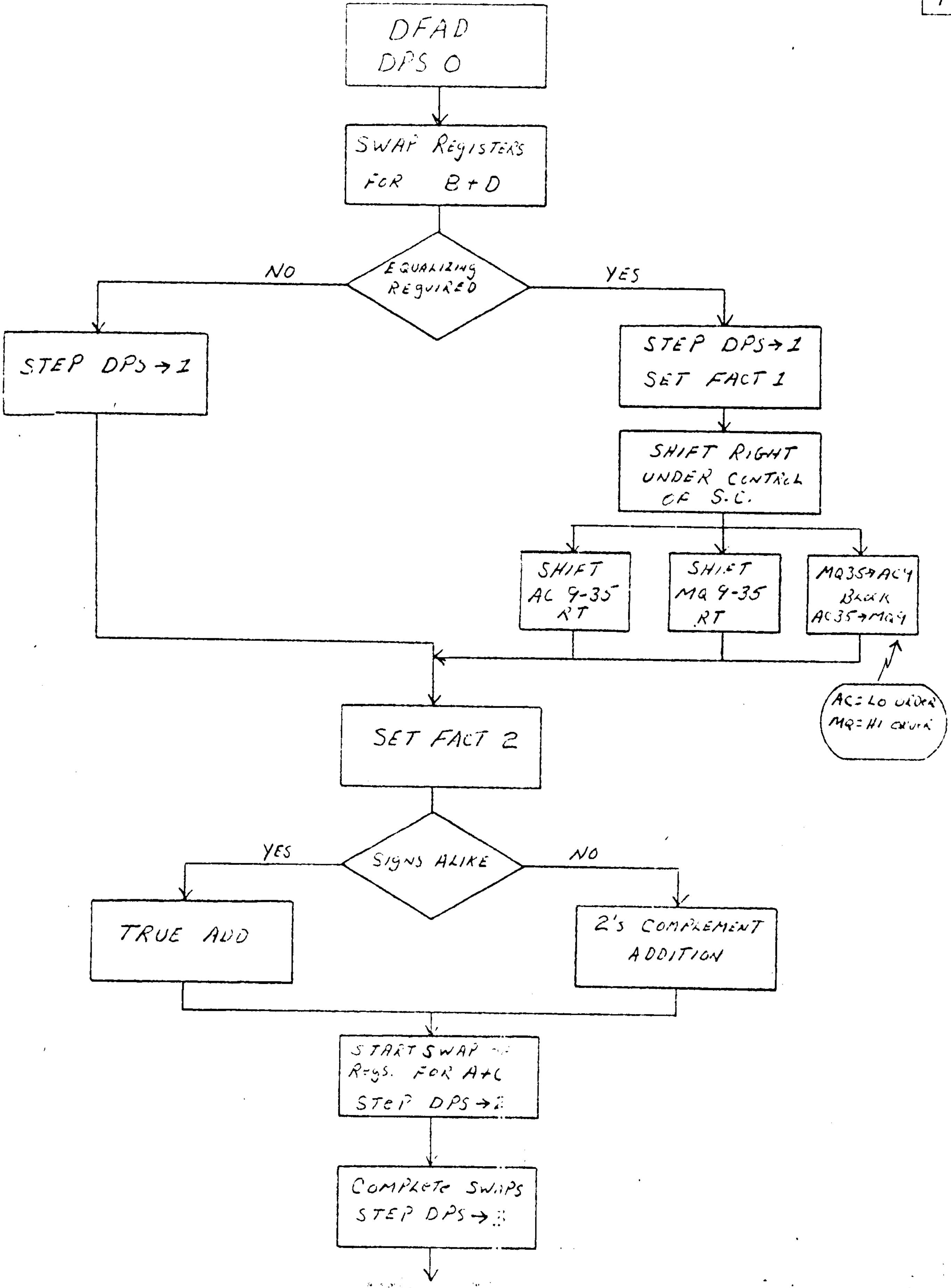


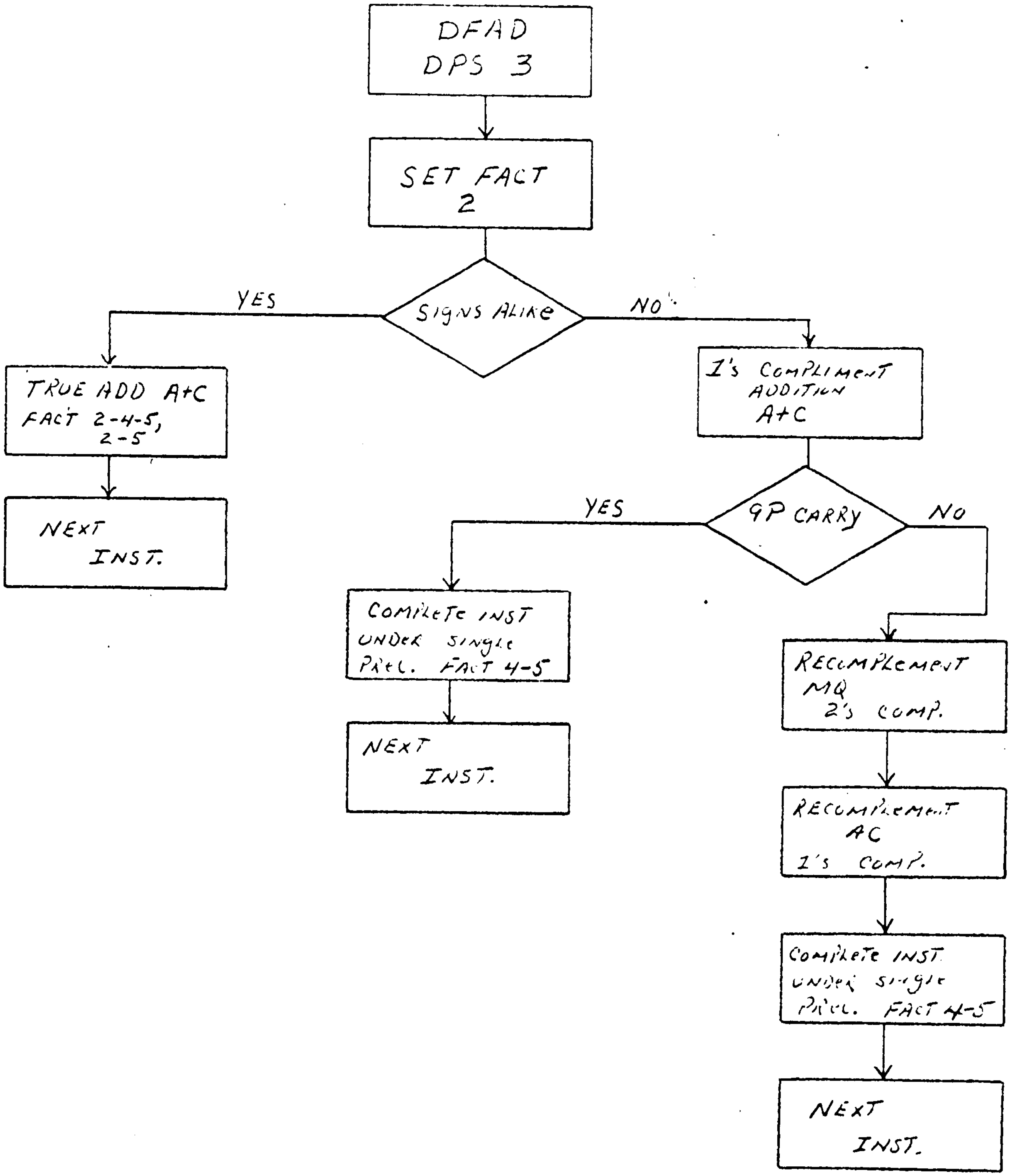
AC = A
MQ = B
SR = C
IB = D

DPS 1 = FACT 1-2,
FACT 2
REMEMBER CARRY FOR DPS 3

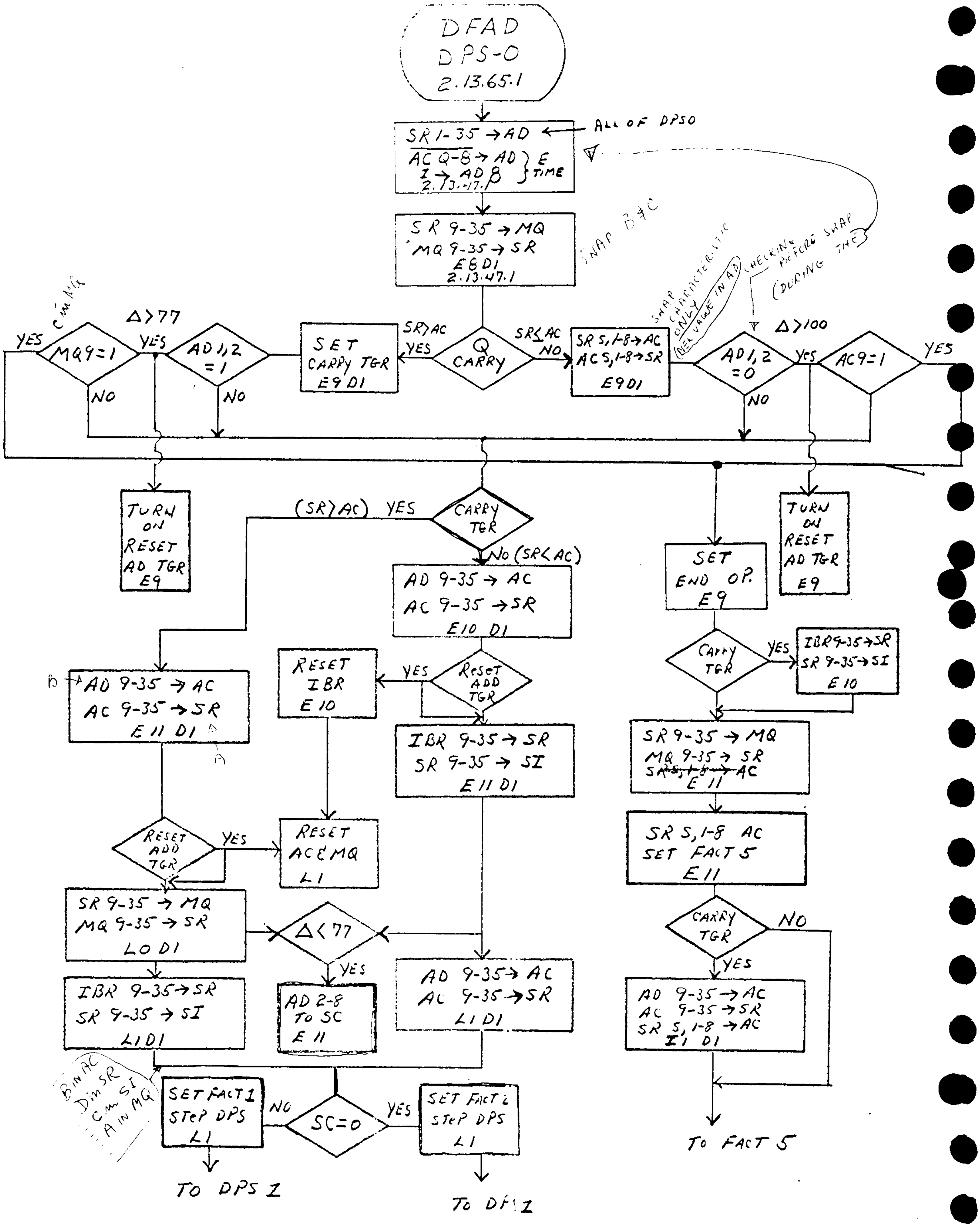
DPS 3 = FACT 2-4-5,
FACT 2-5

DFAD- SIMPLIFIED FLOW CHART

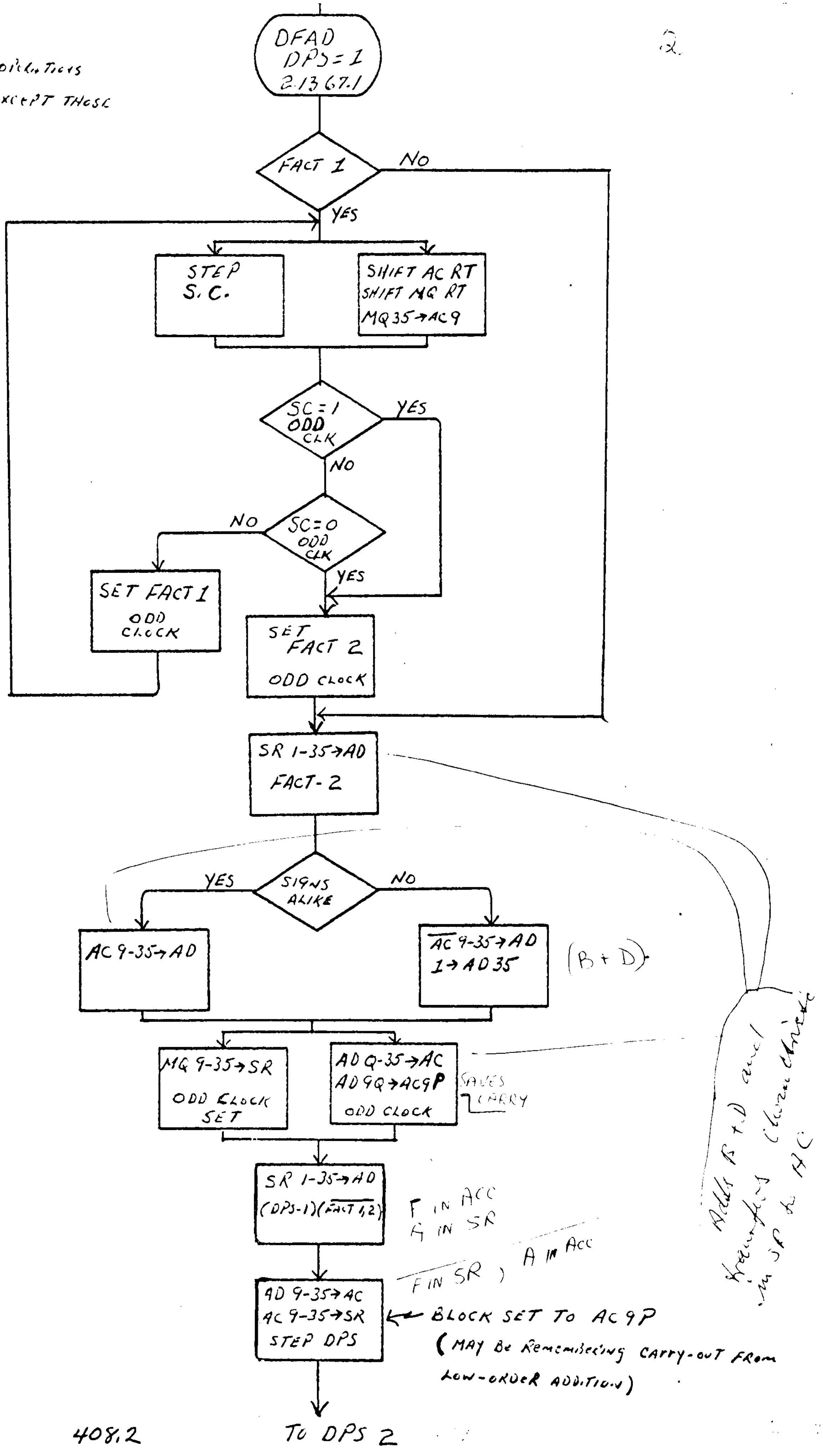


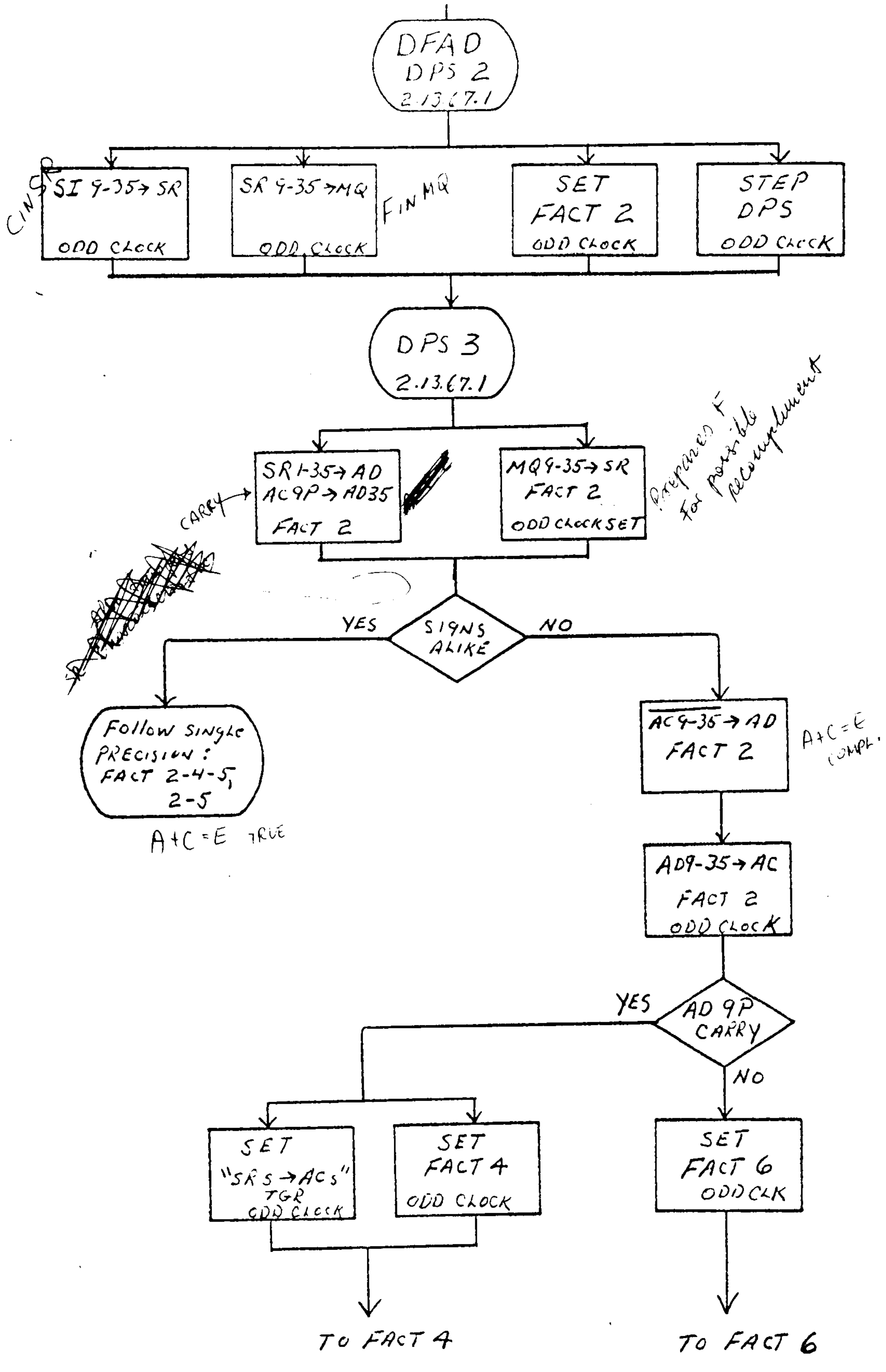

DFAD
DPS 3
SET FACT
2
SIGNS ALIKE
YES
NO
TRUE ADD A+C
FACT 2-4-5,
2-5
NEXT
INST.
1's COMPLIMENT
ADDITION
A+C
9P CARRY
YES
NO
COMPLETE INST
UNDER SINGLE
PREC. FACT 4-5
NEXT
INST.
RECOMPLEMENT
MQ
2's COMP.
RECOMPLEMENT
AC
1's COMP.
COMPLETE INST
UNDER SINGLE
PREC. FACT 4-5
NEXT
INST.

K

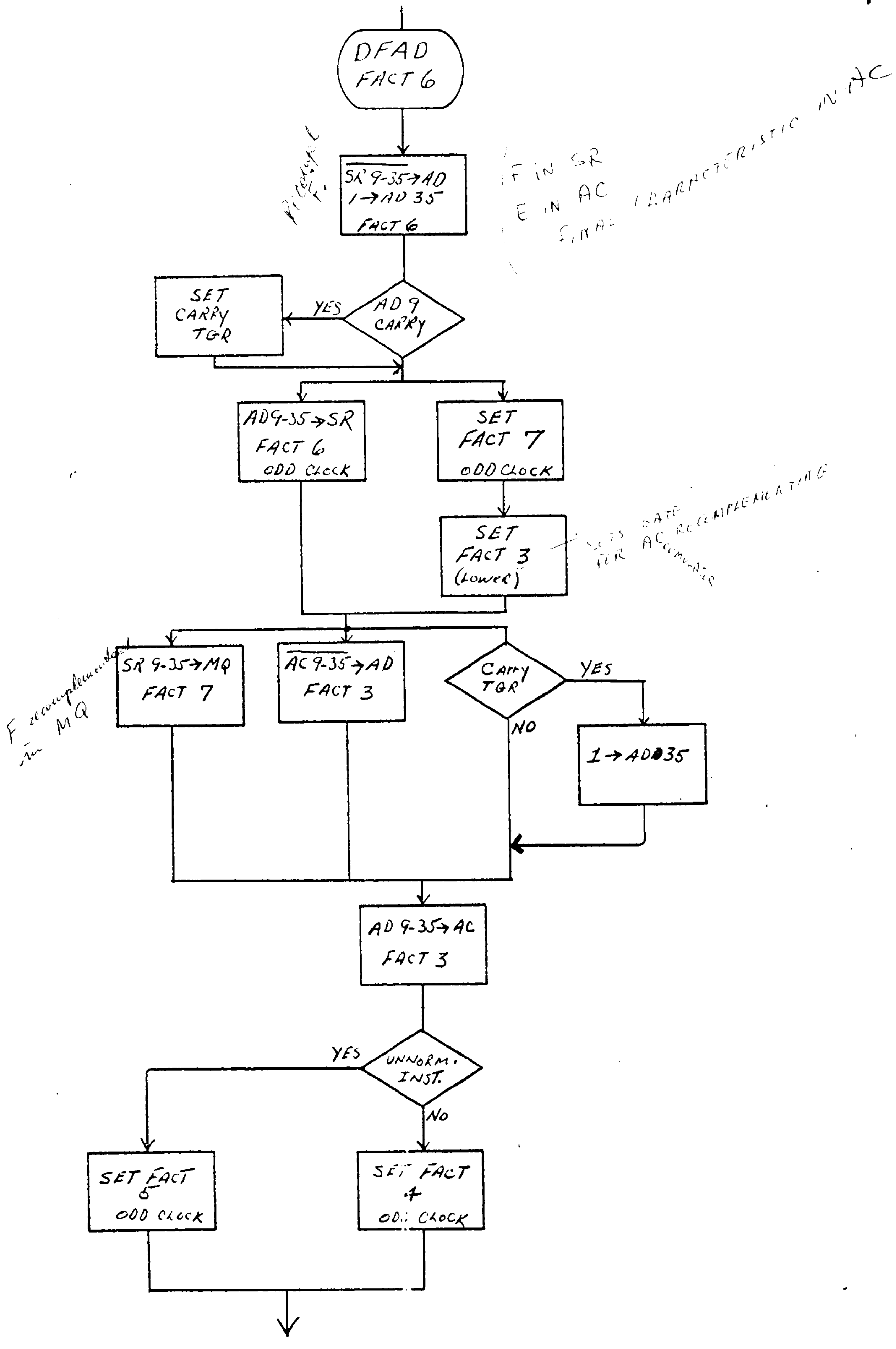

DFAD
DPS-0
2.13.65.1
SR 1-35 → AD
ALL OF DPSO
AC Q-8 → AD
1 → AD 8
2.13.47.1
E TIME
SR 9-35 → MQ
MQ 9-35 → SR
E8 D1
2.13.47.1
SWAP B & C
SWAP CHARACTERISTIC ONLY (NEG VALUE IN AD)
CHECKING BEFORE SWAP (DURING THE
C in MQ
YES
MQ9=1
NO
Δ>77
YES
AD 1,2 =1
NO
SET CARRY TGR E9 D1
SR>AC YES
Q CARRY
SR≤AC NO
SR S,1-8 → AC
AC S,1-8 → SR
E9 D1
Δ>100
AD 1,2 =0
NO
YES
AC9=1
YES
TURN ON RESET AD TGR E9
(SR>AC) YES
CARRY TGR
NO (SR<AC)
AD 9-35 → AC
AC 9-35 → SR
E10 D1
SET END OP. E9
TURN ON RESET AD TGR E9
Carry TGR
YES
IBR 9-35 → SR
SR 9-35 → SI
E 10
RESET IBR E 10
YES
Reset ADD TGR
B
AD 9-35 → AC
AC 9-35 → SR
E11 D1
IBR 9-35 → SR
SR 9-35 → SI
E11 D1
SR 9-35 → MQ
MQ 9-35 → SR
~~SR S,1-8 → AC~~
E 11
SR S,1-8 AC
SET FACT 5
E11
RESET ADD TGR
YES
RESET AC & MQ L1
SR 9-35 → MQ
MQ 9-35 → SR
L0 D1
Δ<77
YES
AD 2-8 TO SC E 11
CARRY TGR
NO
YES
AD 9-35 → AC
AC 9-35 → SR
L1 D1
IBR 9-35 → SR
SR 9-35 → SI
L1 D1
AD 9-35 → AC
AC 9-35 → SR
SR S,1-8 → AC
I1 D1
B in AC
D in SR
C in SI
A in MQ
SET FACT 1
STEP DPS
L1
NO
SC=0
YES
SET FACT 2
STEP DPS
L1
TO DPS 1
TO DPS 1
TO FACT 5

NOTE: ALL FACT 1,2 OPERATIONS ARE BLOCKED EXCEPT THOSE INDICATED.





NOTE: ALL SINGLE PRECISION
FACT 2 DECISIONS FOR
SIGNS UNLIKE ARE BLOCKED.



Follow Single Precision

DEAD REG GATING
NORMAL

3

Q

C≥A

	AC	MQ	SR	SI	IB
6	A	B	-	-	-
7	A	B	C	-	-
8	A	C	B	-	-
TRAP -9	A	C	B	-	D
10	A	C	B	-	D
11	B	C	A	-	D
0	B	A	C	-	D
1	B	A	D	C	D
2					

$\overline{Q}$

C<A

	AC	MQ	SR	SI	IB
6	A	B	-	-	-
7	A	B	C	-	-
8	A	C	B	-	-
9	A	C	B	-	D
10	B	C	A	-	D
11	B	C	D	A	D
0	B	C	D	A	D
1	D	C	B	A	D
2					

E END OP

NORMAL NUMBERS

2

Q

C≥A

CHAR DIFF MORE THAN 77₈ ← B GOOD

	AC	MQ	SR	SI	IB
6	A	B	-	-	-
7	A	B	C	-	-
8	A	C	B	-	-
9	A	C	B	-	D
10	A	C	D	B	D
11	A	D	C	B	D
0	A	D	C	B	D
1	C	D	A	B	D
2					

$\overline{Q}$

C<A

CHAR DIFF MORE THAN 77₈

	AC	MQ	SR	SI	IB
6	A	B	-	-	-
7	A	B	C	-	-
8	A	C	B	-	-
9	A	C	B	-	D
10	A	C	B	-	D
11	A	B	C	-	D
0	A	B	C	-	D
1	A	B	C	-	D
2					

P408.5

408. Double Precision Floating Point Multiply

As stated in the beginning of this section, the purpose of floating point arithmetic is the handling of very large or very small numbers rapidly and accurately. Double precision further extends this capability by doubling the fraction-handling capacity, thereby doubling the precision of the numbers involved. The end result of a double precision floating point number is a product consisting of characteristic and fraction, the fraction being 54 bits long. This range enables programmers to work with numbers as small as 2^{-53}. Such a number would have the following format:

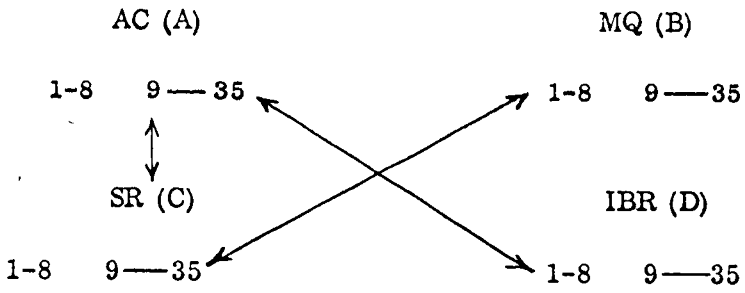
.0000000000000000000000000000000000000000000000000000000000001

The most significant 27 bits of the answer are contained in the AC and the least significant in the MQ. The characteristic is contained in positions 1-8 of the AC and this number (-27), is repeated in the first eight positions of the MQ.

When using double precision floating point numbers the largest of the two numbers must be in the even-numbered core storage location and the smaller number in the next higher odd location. Also, the largest fraction of the multiplier must be in the AC and the smaller fraction in the MQ (this number would have been brought into the computer by a double load instruction).

The computer would then perform three multiplications and two additions and place the sum in the AC and MQ.

Multiplication is performed as follows:



where

$AC = A^{n}$

$MQ = B^{n-27}$

$SR = C^{m}$

$IBR = D^{cm-27}$

and the addition is performed as follows:

$$\begin{array}{l} A^{n} + B^{n} - 27 \\ C^{m} + D^{m} - 27 \\ \hline AC^{m+n} + (CB + AD)^{(m+n)-27} \end{array}$$

For convenience, ~~bits~~ let's regroup the numbers into characteristic and fraction:

where

$A^{n} = A^{\text{characteristic}}\ A^{\text{fraction}}$

$B = B^{\text{characteristic}}\ B^{\text{fraction}}$

$C = C^{\text{characteristic}}\ C^{\text{fraction}}$

$D = D^{\text{characteristic}}\ D^{\text{fraction}}$

In performing the multiplication the computer will add the characteristics of A and C and multiply all four fractions. Two distinct fractions are apparent - processing the characteristic and proc~~urr~~essing the fractions.

Following is the number handling sequence in the 7094 during a double precision floating point multiply instruction:

1. $\begin{array}{rl} A & \text{characteristic} \\ +\ C & \text{characteristic} \\ \hline A{+}C & \text{characteristic} \end{array} \qquad \begin{array}{rl} B & \text{fraction} \\ \times\ C & \text{fraction} \\ \hline BC & \text{fraction} \end{array}$

2. $\begin{array}{rl} A & \text{fraction} \\ \times\ D & \text{fraction} \\ \hline AD & \text{fraction} \end{array}$

3. $\begin{array}{rl} AD & \text{fraction} \\ +\ BC & \text{fraction} \\ \hline AD{+}BC & \text{fraction} \end{array}$

4. $\begin{array}{rl} A & \text{fraction} \\ \times\ C & \text{fraction} \\ \hline AC & \text{fraction} \end{array}$

5. $A{+}C\ \text{characteristic} \qquad \begin{array}{rl} AD + BC & \text{fraction} \\ +\ AC & \text{fraction} \\ \hline AD{+}AC{+}BC & \text{fraction} \end{array}$

Only the most significant 27 bits of BC$^{\text{fraction}}$ and AD$^{\text{fraction}}$ are retained. These are then added to the least significant bits of AC$^{\text{fraction}}$.

Note the first step shows simultaneous operation. This is possible because we are using one of the low order portions of the multiplicand, i.e., the fraction in the MQ. No carry out of this operation is possible and the A$^{\text{chqracteristic}}$ and B$^{\text{characteristic}}$ can be added and temporarily held pending completion of the problem

Much general housekeeping and checking must be done during the processing of a

DPFM instruction. The computer uses the following five registers during DPFM:

AC
MQ
SR
SI
IBR

The AC - MQ combination contains the original multiplier and the SR - IBR combination contain the multiplicand. The SI is used primarily for register swapping and temporary storage.

This procedure can be demonstrated by a small decimal problem. Assume the problem is to multiply 2.57 by 3.42. This calls for a simple multiplication of the multiplicand by each number in the multiplier:

```
  2.57
    3.42
 ---------
    5 14
  1 02 8
  7 71
 ---------
  8.78 94
```

The computer handles the problem slightly different, by performing three multiplications (as we did above) and then two additions. The term:

$2.57 = \overset{A}{2.50} + \overset{B}{.07}$ (to correspond to AC - MQ)

$3.42 = \overset{C}{3.40} + \overset{D}{.02}$ (to correspond to SR - IBR)

the problem will be performed by:

$$\begin{array}{ll} 1) & B \cdot C = X \\ 2) & A \cdot D = \underline{+ Y} \\ & Z \end{array}$$

3) $A \cdot C + Z = \text{Sum}$

The first step (B • C) yields

1. 3.40 • .07 = .2380

the last two digits would be l~~a~~ost, giving .23. This is saved.

2. The second step (A• D) yields,

   2.50 • .02 = .0500

The .05 of this is added to the first partial product and the sum is .28.

3. Step three (A • C) yields,

   2.5 • 3.4 = 8.50.

All of the fractions are added and the final answer = 8.78, which compares with the answer obtained by the longhand method - 8.7894. Some accuracy was lost because of dropping two digits but this would not occur in the computer. Notice that we did not add the 2 and 3 - but multiplied them since they are true numbers and not characteristic expressions.

~~7094 DR~~FM

~~Computer time can be divided into E and L times, with E time being used for register alignment, characteristic addition and L time used for fraction multiplication.~~

408.24

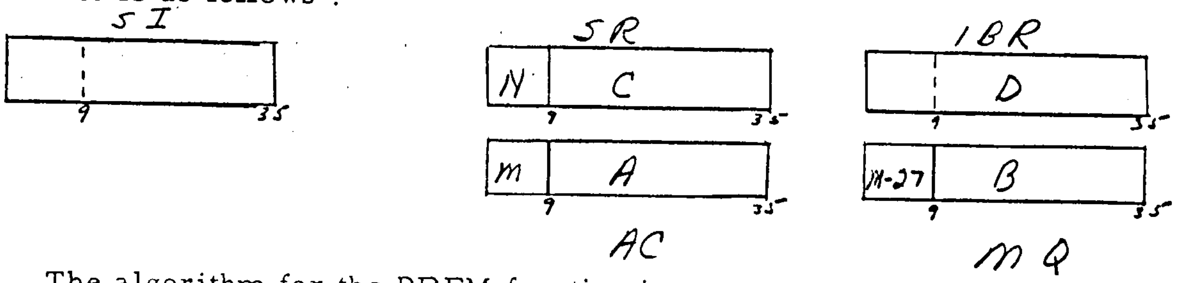
7094 DPFM

A DPFM instruction to the 7094 results in multiplying the contents of storage location Y and Y + 1 by the contents of the AC and MQ. Five registers are used in executing this instruction; the fifth (sense indicator) being used primarily for temporary storage during swapping and multiply iterations.

The following discussion pertains to the flow chart on page 408. 40 which can be briefly summarized in the steps below :

1. Add characteristics of the two floating point operands.
2. Multiply Large fraction of Y by small fraction in MQ. $(B \cdot C)$
3. Multiply Large fraction in AC by small fraction of Y + 1. $(A \cdot D)$
4. Add the products of steps 2 and 3. $(B \cdot C + A \cdot D)$
5. Multiply the large fraction of Y by the large fraction in the AC. $(A \cdot C)$
6. Add the product of step 5 to the sum of step 4. $[A \cdot C + (B \cdot C + A \cdot D)]$
7. Adjust MQ characteristic.

The format at the start of the DPFM instruction, showing register contents is as follows :



The algorithm for the DPFM function is :

$$(A^{m} + B^{m-27})(C^{n} + D^{n-27}) = AC^{m+n} + (BC + AD)^{m+n-27} + (BD)^{m+n-54}$$

In performing a DPFM the computer will first multiply B x C, then A x D and finally, A x C. The three products are added and the final sum placed in the AC and MQ ;with the AC containing the characteristic and the 27 most significant bits of the answer; and the MQ containing the characteristic - 27_{10} and the 27 least significant bits of the final answer. This double precision number may be stored as such or it may be used in the next computation the computer performs.

Recall from previous discussions on floating point that the small order number of the two operands contains a characteristic that is 27_{10} less than the characteristic of the high order number or data word. Although this characteristic will be brought out of core storage as part of the second word, the characteristic is ignored by the computer. The same is true for the characteristic in the MQ at the beginning of the DPFM instruction. For this reason, the dissussion on the flow chart will not attempt to keep track of these two characteristics. The characteristics of the high order numbers are added early in the DPFM sequence and once in the AC the sum is not disturbed until the end of the instruction when it is adjusted by -27_{10} and placed in the MQ.

E Time of DPFM

The following steps are performed during E time of a DPFM instruction :

1. Set SC to 33_8

2. SB to SR -- brings in the contents of storage loaction Y and places the contents in the storage register.

3. SR 1-8 (A) to AD ~~1 - 8~~ -- done all during E time to add the characteristics of the SR and AC. This will be the characteris;tic of our final answer.

4. AC to AD, 1 - 8 -- brings the characteristic of A (m) to the AD to be added to the characteristic in the SR, the characteristic of C (n).

5. Zero checks of MQ and SB -- if either the MQ or the SB (SR data) is zero, the instruction is terminated.

6. Reset MQ -- this is done so the characteristics won't be added twice.

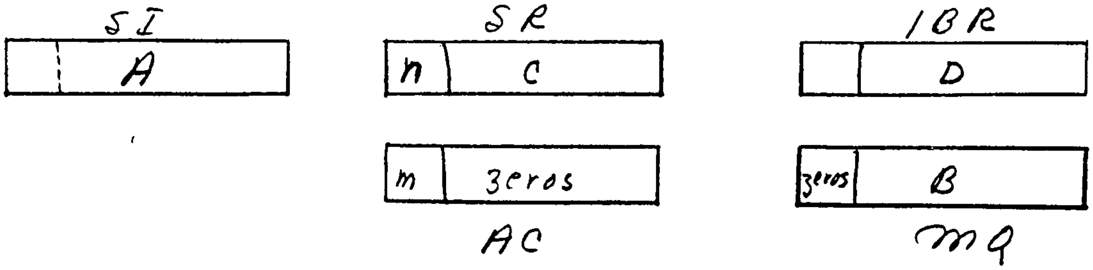
7. SR 9 - 35 to SI -- stores fraction of C in the SI.

8. AC 9-35 to SR -- puts fraction of A in the SR.

9. D read from SB into IBR.

10. AD 9 - 35 to AC -- this resets AC 9 - 35 in preparation to receive the results of the first multiply.

11. SR and SI swapped -- puts A in the SI and C in the SR. At this point the registers contain :



12. Decode last two bits of MQ ;in preparation for the first multiply iteration.

At the end of E time the registers contain :

SI -- A (9 - 35)

SR -- C (9 - 35) and n (1 - 8)

IBR -- D (9 - 35) and n-27 in 1 - 8.

AC -- n + m in 1 - 8 and zeros in 9 - 35

MQ -- zeros in 1 - 8 and B in 9 - 35.

Bits 34 and 35 of the MQ have been decoded and the appropriate multiply trigger set.

The sum of n + m in the AC is too large by 200_8 because both characteristics contained this number. One of the first steps taken in L time will be to subtract 200_8 from the sum of the characteristics.

The computer will now go to L time for the multiply cycle -- refer to the discussion presented in MPY for a description of the multiply cycle.

The objectives for the first L time include:

1. Continue mulitply iterations until SC = 0

2. Subtract 200_8 from AC 1 - 8.

Upon completion of the first multiply the registers contain :

SI -- A in 9 - 35

SR -- C in 9 - 35

IBR -- D in 9 - 35

AC -- n + m in 1 - 8 and the most significant portion of B x C in 9 - 35.

MQ -- least significant part of B x C.

Second Multiply

The next stage in processing the DPFM instruction requires the multiplication of A x D. Before this can be carried out, the registers must be realigned until the SR contains A and the MQ contains D; Also, the results of the previous multiply, B x C, must be temporarily stored. The following steps are performed during the second multiply :

1. Set SC to 33_8.

2. IBR to SR -- puts D into the SR

3. SR to MQ -- puts C into the MQ and destroys the least significant bits of B x C.

4. AC to I BR -- this stores the results of B x C in the IBR. Although n + m will also be transferred in 1 - 8, the characteristic is also left in the AC. IBR 1 - 8 will never be used.

5. MQ to SR -- puts C in the SR

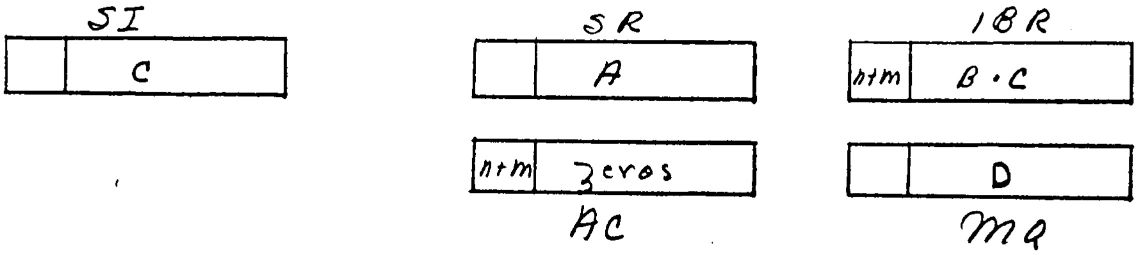
6. SR to MQ -- this puts D into the MQ.

7. SR to SI -- this puts A into the SR

8. SI to SR -- this puts C in the SI

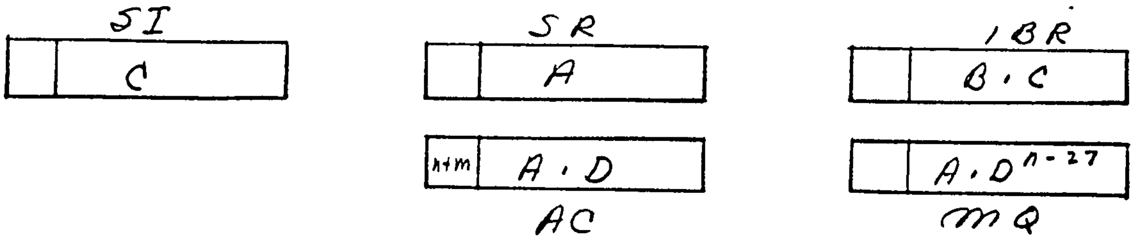
9. Reset AC 9 - 35 preparatory to accepting the results of A x D.

At the end of the register swapping and resets the registers contain :



10. Turn on MPY cycle trigger.

11. Continue multiply iterations until SC = 0. When SC = 0 the AC contains the most sigñificant portion of A x D and the MQ contains the least significant. At the end of the second multiply the registers look like this :



Third Multiply

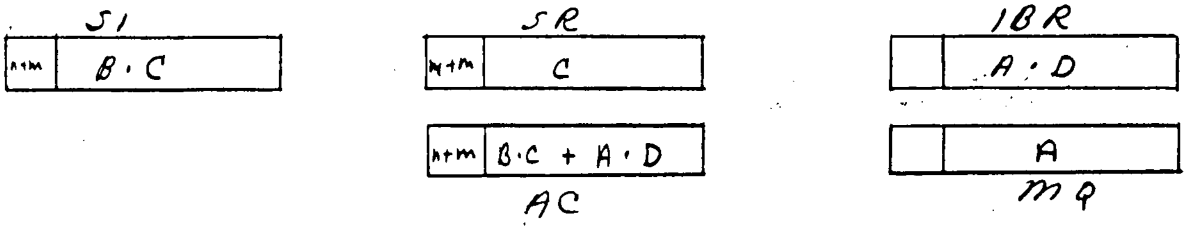
Recall that the algorithm of the DPFM shows three multiplies and two additions are necessary to complete the DPFM ($\underset{mpy}{A}C^{m+n} \underset{add}{+} (\underset{mpy}{BC} \underset{add}{+} \underset{mpy}{AD})^{m+n-27}$).
Note - one more add for characteristics.

During the third and final multiply the product of A x C will be developed as well as the sum of B x C and A x D. Upon completeion of the second multiply the double precision sync (DPS) trigger was stepped and now DPS ≠ 0. Note on page three of the flow chart that when DPS $\neq 0$, an addition is performed while register swapping is taking place for the final multiply. This addition places the sum of B x C and A x D in the AC. This sum is left there during the multiply cycle and,as the product of A x C is developed, addition is performed, adding the least significant bits of A x C to the sum of B x C and A x D. Also, if a carry develops from the fraction addition DPS will be set to 1 and a 1 added to AC 35 to increment the product of A x C.

The following steps are performed during the third multiply :

1. Set SC to 33_8.
2. IBR to SR -- puts B x C in the SR
3. SR to MQ -- puts A into the MQ.
4. AC to IBR -- puts product of A x D into the IBR.
5. SR 9 - 35 to AD -- brings product of B x C to the adders.
6. 1 to AD9P -- this will propagate any fraction addition carry.
7. AC to AD -- brings product of A X D to adders.
8. AD to AC -- puts sum of B x C and A x D into the AC.
9. Turn on pre end op.
10. Step DPS to 1 if there is a fraction carry.
11. SR to SI -- puts B x C into SI (not used any more).
12. SI to SR -- puts C into the SR

At the end of this register swapping the registers contain :



13. Turn on MPY cycle trigger.

14. Continue myp iterations until SC =).

As the product of this final multiply is developed it is placed into the AC which already contains the sum of B x C and A x D. Addition taks place so that the sum of B x C and A x D is modified by the low order bits of A α C. The AC and MQ are shifted right during multiply. Upon completion of the third multiply the high order [redacted] product of A x C is in the AC and the MQ contains B x C plus A x D and may also contain portions of A x C.

The sum of the characteristics, n + m must be modified to n + m - 27_{10} and placed into the MQ.

At the completion of the third multiply the MPY cycle trigger is off and SC = 0 (page 3 of the flow chart). The computer now :

1. Takes AC to AD with a 1 to AD 35 -- this is to pick up the fraction carry if DPS had been stepped. If DPS was not stepped the result is left in the adders. If DPS was stepped the result is taken back to the AC and is the final high-order answer.

2. Turn on FP trigger.

3. Turn on FACT 5 -- this is to adjust the MQ characteristic as shown on page 4 of the flow chart.

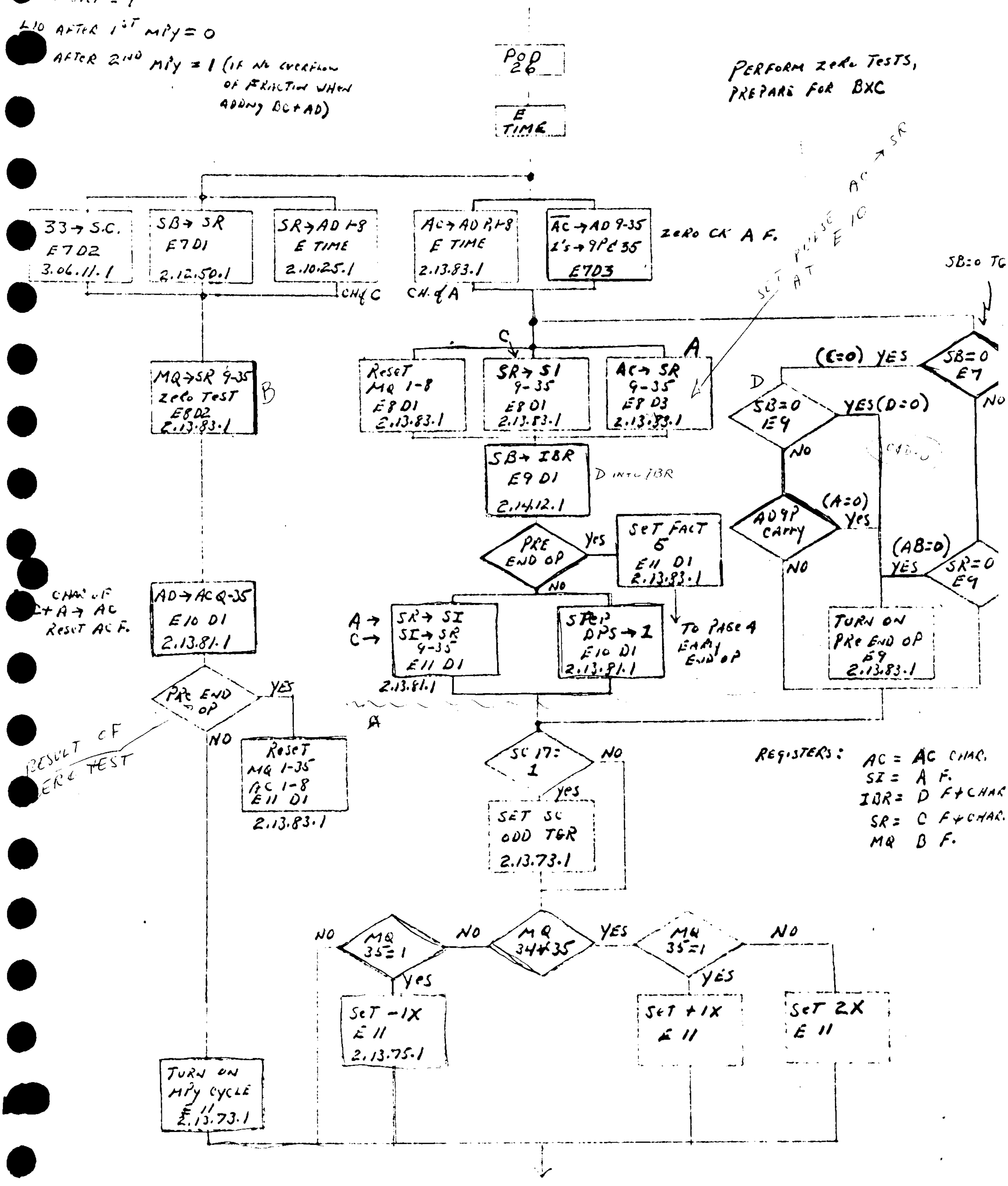
4. Reduce AC characteristic by 27_{10} and place the result in the MQ.

END OP

D.P. FLOATING MPY

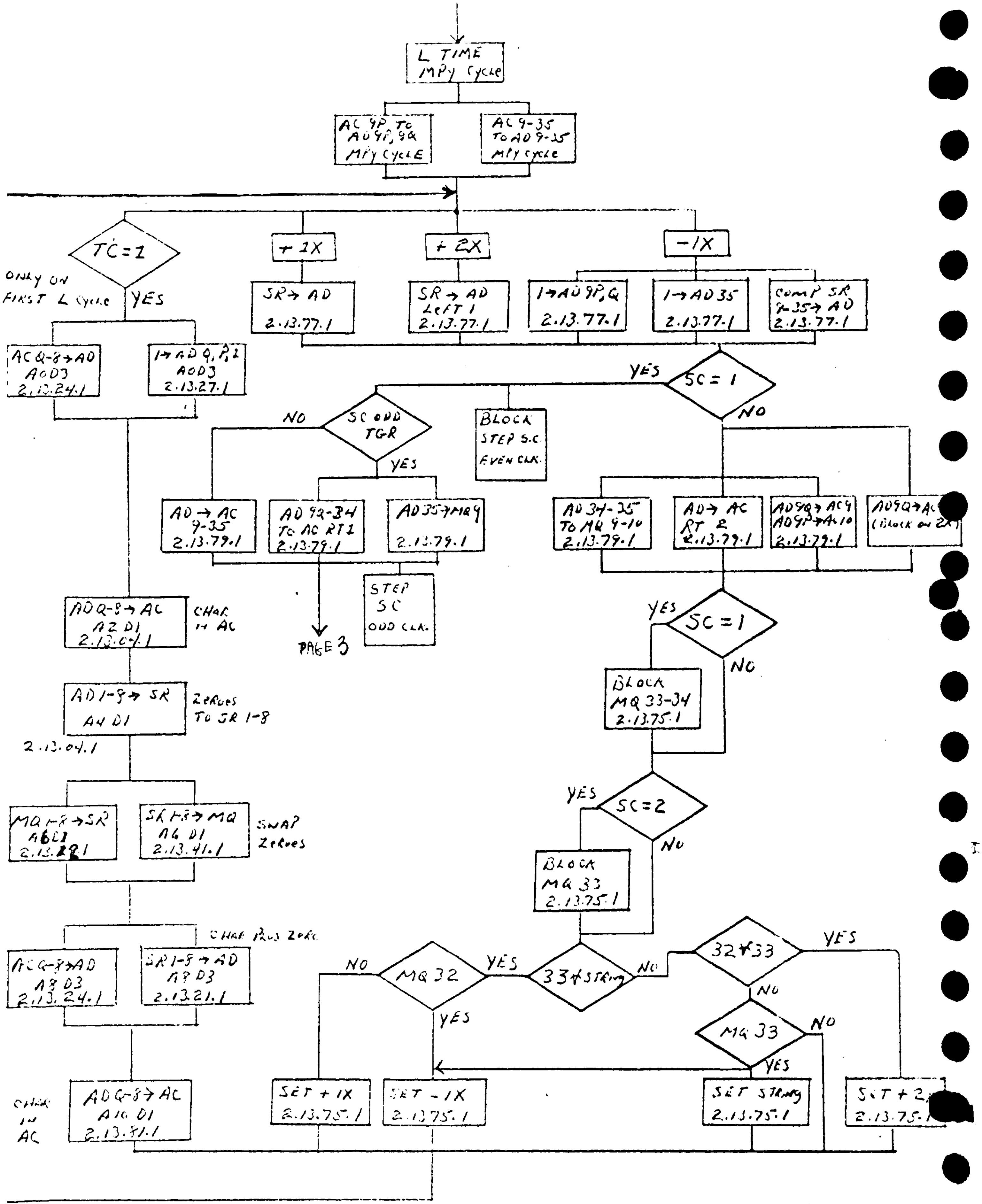
DPS SEQUENCE
I TIME = 0
E 10 STEP = 1
L10 AFTER 1ST MPY = 0
AFTER 2ND MPY = 1 (IF NO OVERFLOW OF FRACTION WHEN ADDING DC+AD)

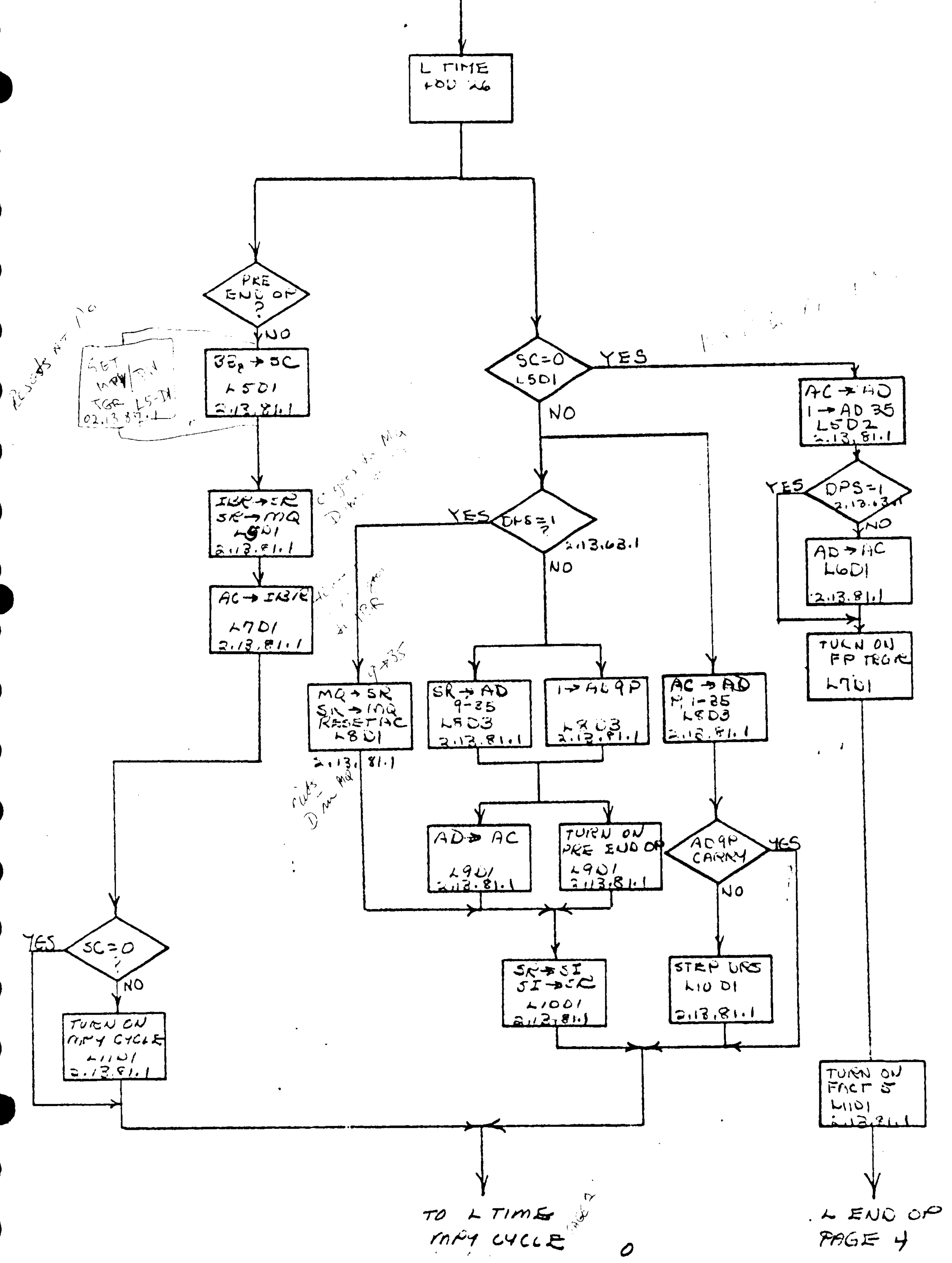
PERFORM ZERO TESTS, PREPARE FOR BXC

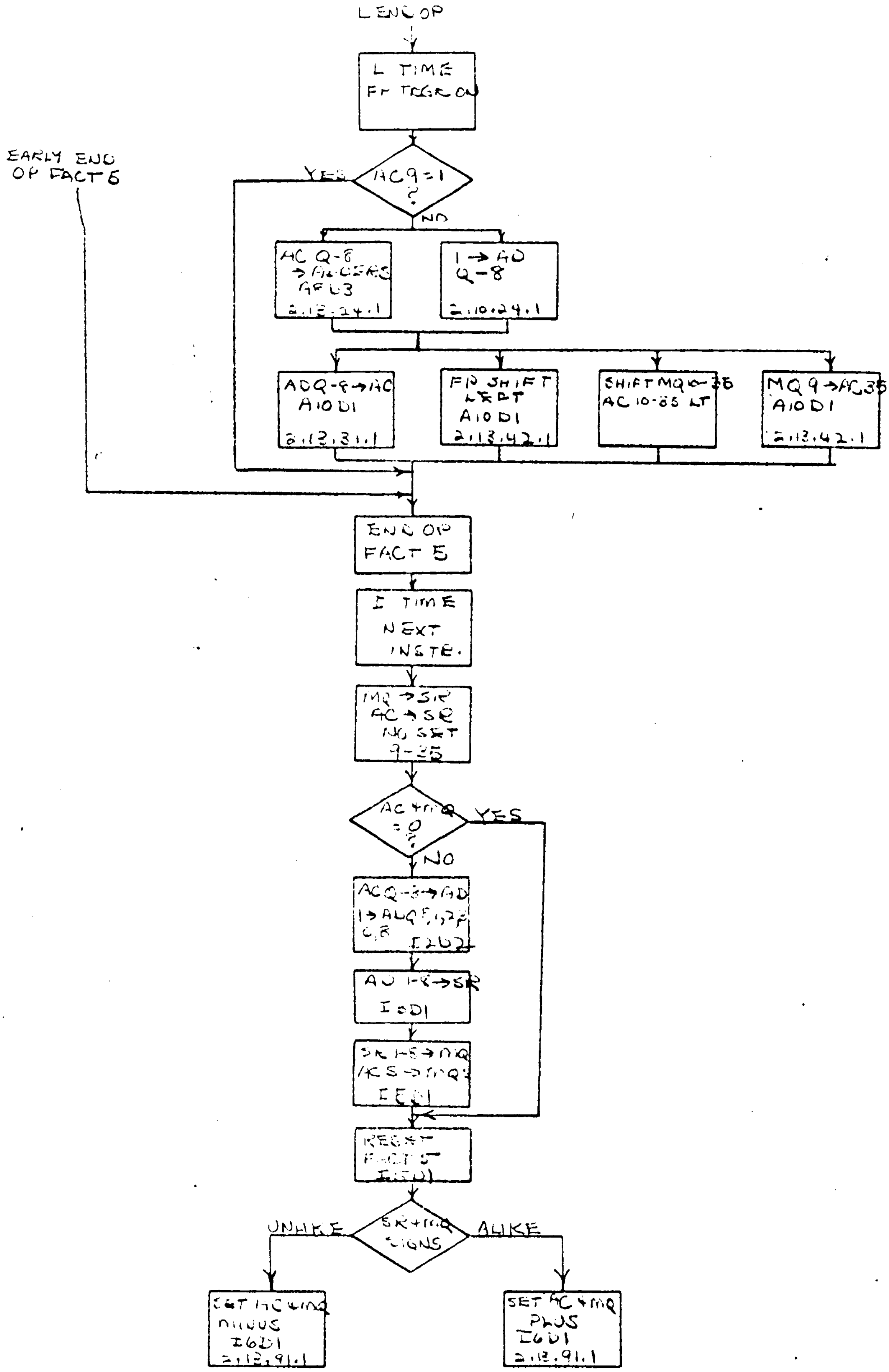

POP 26
E TIME
33 → S.C. E7D2 3.06.11.1
SB → SR E7D1 2.12.50.1
SR → AD 1-8 E TIME 2.10.25.1
CH of C
AC → AD P,1-8 E TIME 2.13.83.1
CH. of A
AC → AD 9-35 1's → 9PE 35 E7D3
ZERO CK A F.
SET PULSE AT E 10 AC → SR
SB:0 TGR
MQ → SR 9-35 ZERO TEST E8D2 2.13.83.1
B
C
A
RESET MQ 1-8 E8 D1 2.13.83.1
SR → SI 9-35 E8 D1 2.13.83.1
AC → SR 9-35 E8 D3 2.13.83.1
SB → IBR E9 D1 2.14.12.1
D INTO IBR
(C=0) YES
SB=0 E7
NO
D
SB=0 E9
YES (D=0)
NO
C+D=0
AD 9P CARRY
(A=0) YES
NO
(AB=0) YES
SR=0 E9
TURN ON PRE END OP E9 2.13.83.1
PRE END OP
YES
NO
SET FACT 5 E11 D1 2.13.83.1
TO PAGE 4 EARLY END OP
CHAR OF C+A → AC RESET AC F.
AD → AC Q-35 E10 D1 2.13.81.1
A →
C →
SR → SI SI → SR 9-35 E11 D1 2.13.81.1
STEP DPS → 1 E10 D1 2.13.81.1
PRE END OP
YES
NO
RESULT OF ZERO TEST
RESET MQ 1-35 AC 1-8 E11 D1 2.13.83.1
A
SC 17: 1
NO
YES
SET SC ODD TGR 2.13.73.1
NO
MQ 35=1
NO
MQ 34≠35
YES
MQ 35=1
NO
YES
YES
SET -1X E11 2.13.75.1
SET +1X E11
SET 2X E11
TURN ON MPY CYCLE E11 2.13.73.1
TO PAGE 2 L TIME

REGISTERS:
AC = AC CHAR.
SI = A F.
IBR = D F+CHAR
SR = C F+CHAR.
MQ B F.

408,40


L TIME
MPy Cycle
AC 9P, TO AD 9P, 9Q MPy Cycle
AC 9-35 TO AD 9-35 MPy cycle
TC = 1
ONLY ON FIRST L Cycle
YES
+ 1X
+ 2X
- 1X
SR → AD 2.13.77.1
SR → AD Left 1 2.13.77.1
1 → AD 9P, Q 2.13.77.1
1 → AD 35 2.13.77.1
COMP SR 9-35 → AD 2.13.77.1
ACQ-8 → AD A0D3 2.13.24.1
1 → AD Q, P, 1 A0D3 2.13.27.1
YES
SC = 1
NO
NO
SC ODD TGR
YES
BLOCK STEP S.C. EVEN CLK.
AD → AC 9-35 2.13.79.1
AD 9Q-34 To AC RT 1 2.13.79.1
AD 35 → MQ 9 2.13.79.1
STEP SC ODD CLK.
PAGE 3
AD 34-35 TO MQ 9-10 2.13.79.1
AD → AC RT 2 2.13.79.1
AD 9Q → AC 9 AD 9P → AC 10 2.13.79.1
AD 9Q → AC 9 (Block on 2X)
ADQ-8 → AC A2 D1 2.13.04.1
CHAR IN AC
AD 1-8 → SR A4 D1
ZEROES TO SR 1-8
2.13.04.1
YES
SC = 1
NO
BLOCK MQ 33-34 2.13.75.1
MQ 1-8 → SR A6 D1 2.13.21.1
SR 1-8 → MQ A6 D1 2.13.41.1
SWAP ZEROES
YES
SC = 2
NO
BLOCK MQ 33 2.13.75.1
CHAR PLUS ZERO
ACQ-8 → AD A8 D3 2.13.24.1
SR 1-8 → AD A8 D3 2.13.21.1
NO
MQ 32
YES
33 + STRING
NO
32 + 33
YES
YES
NO
MQ 33
NO
YES
SET + 1X 2.13.75.1
SET - 1X 2.13.75.1
SET STRING 2.13.75.1
SET + 2X 2.13.75.1
CHAR IN AC
ADQ-8 → AC A10 D1 2.13.31.1


L TIME
FOU 26
PRE
END OP
?
NO
$3E_8$ → SC
L5D1
2.13.81.1
SET
MPY/DIV
TGR L5-D1
02.13.87.1
Resets at No
IBR → SR
SR → MQ
L6D1
2.13.81.1
AC → IB/R
L7D1
2.13.81.1
SC=0
L5D1
YES
NO
AC → AD
1 → AD 35
L5D2
2.13.81.1
DPS=1
2.13.63.1
YES
NO
AD → AC
L6D1
2.13.81.1
TURN ON
FP TRGR
L7D1
DPS=1
?
2.13.63.1
YES
NO
MQ → SR
SR → MQ
RESET AC
L8D1
2.13.81.1
9→35
SR → AD
9-35
L8D3
2.13.81.1
1 → AD 9P
L8D3
2.13.81.1
AC → AD
P, 1-35
L8D3
2.13.81.1
AD → AC
L9D1
2.13.81.1
TURN ON
PRE END OP
L9D1
2.13.81.1
AD 9P
CARRY
YES
NO
SR → SI
SI → SR
L10D1
2.13.81.1
STEP URS
L10 D1
2.13.81.1
SC=0
?
YES
NO
TURN ON
MPY CYCLE
L11D1
2.13.81.1
TURN ON
FACT 5
L11D1
2.13.81.1
TO L TIME
MPY CYCLE
page 2
L END OP
PAGE 4


L END OP
L TIME
FP TRGR ON
EARLY END
OP FACT 5
YES
AC9=1
?
NO
AC Q-8
→ AC CERS
AFL3
2.12.24.1
1 → AD
Q-8
2.10.24.1
ADQ-8→AC
A10DI
2.13.31.1
FP SHIFT
LEFT
A10DI
2.13.42.1
SHIFT MQ 10-35
AC 10-35 LT
MQ 9→AC35
A10DI
2.13.42.1
END OP
FACT 5
I TIME
NEXT
INSTR.
MQ→SR
AC→SE
NO SET
9-35
AC+MQ
=0
?
YES
NO
ACQ-8→AD
1→ALQF, 12F
6, 8
I2D2
AD 1-8→SR
I5D1
SR 1-5→MQ
ACS→MQ
I5D1
RESET
FACT 5
I5D1
UNLIKE
SR+MQ
SIGNS
ALIKE
SET AC+MQ
MINUS
I6D1
2.13.91.1
SET AC+MQ
PLUS
I6D1
2.13.91.1

DOUBLE PRECISION FLOATING POINT DIVIDE
(DPFD) (Flow Chart on 409.10)

A double precision floating point divide instruction to the 7094 results in dividing the double precision floating point number in the AC and MQ by a double precision floating point number in core storage locations Y (even numbered location) and Y + 1 (odd numbered location). The answer appears as a double precision floating point number in the AC and MQ. The AC contains $Q_1 \cdot 2^{n-m}$, the high order bits of the quotient. The MQ contains $Q_2 \cdot 2^{n-m-27}$, the low order bits of the quotient.

The algorithm for the double precision floating point divide function is:

$$\frac{A \cdot 2^n + B \cdot 2^{n-27}}{C \cdot 2^m + D \cdot 2^{m-27}} = Q_1 + Q_2$$

Let:

$$A = A \cdot 2^n$$
$$B = B \cdot 2^{n-27}$$
$$C = C \cdot 2^m$$
$$D = D \cdot 2^{m-27}$$

Then:

$$\frac{A + B}{C + D} = Q_1 + Q_2$$

Where:

$$Q_1 + R_1 \text{ (remainder)} = \frac{A + B}{C}$$

$$Q_2 = \frac{R_1 - Q_1 D}{C}$$

The sequential steps taken by the computer in processing the DPFD instruction includes the followir g:

1. Divide $\frac{A + B}{C} = Q_1 + R_1$

2. Multiply $Q_1 \times D = Q_1 D$

3. Add $\overline{R_1} + Q_1D$

4. Divide $\dfrac{\overline{R_1} + Q_1D}{C} = Q_2$

5. Add $Q_1 + Q_2$

As can be seen from the above, the processing of this instruction requires single precision division, multiplication, subtraction and addition. In fact, all of the arithmetic capabilities of the computer are used in processing a DPFD instruction.

E Time

During E time the following events take place:

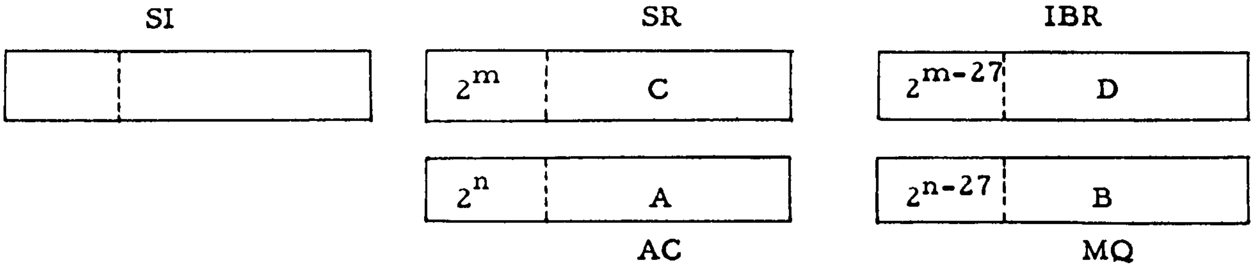
1. Y to SR.

2. Y + 1 to IBR.

3. Set SC to 33_8.

4. Divide AC and MQ by two. This is accomplished by shifting the AC and MQ right one place. AC 35 is saved by shifting to MQ 9. MQ35 is saved by ring shifting it to MQS.

At the end of E time the registers contain:



L Time First Step

The objectives for first step L time are: (See single precision floating point divide flow chart).

1. Divide check test.

2. Set MQ sign.

3. End op if AC and MQ fractions = 0.

4. Put characteristic difference in AC if not divide check.

The divide check test determines if the dividend is greater than twice the value of the divisor. The AC was divided by two during E time. The new value of the AC (one-half the original) is complemented and added to the contents of the storage register. No carry out of bit 9 means the dividend is greater than the divisor by twice as much and the divide check trigger is turned on. If the dividend is more than twice as great as the divisor, this means the quotient will be greater than two. Because we are dealing with fractions, the whole number value of two would exceed register capacity.

With the divide check trigger on, the instruction is terminated during this L cycle. The divide check trigger cannot be turned on if the divisor (at least) is normalized. If the floating point numbers are normalized to begin with the answer will be normalized.

The AC and MQ are shifted left one place (MQ 9 to AC 35 and MQ S to MQ 35) to restore the dividend to normal.

MQ sign is set following comparison of the signs of the AC and SR. If signs are alike, MQ S is reset (0). If signs are unlike, MQ S is set to 1 (minus).

The zero check of the AC is made by complementing AC Q - 35 to the adders and adding 1 to AD 35. A carry out of bit 9 indicates AC = 0 and the column 9 carry trigger is turned on. Tha column 9 carry trigger will terminate the instruction later in the cycle if MQ = 0. The contents of SR 1 - 8 are brought at the same time as AC. The result of this subtraction is the characteristic difference between the AC characteristic and the SR characteristic (n-m). This characteristic difference is returned to the AC. The value is incorrect by 200_8 (recall this is the 0 value between positive and negative numbers) because both numbers carried 200_8 and the value was lost during the subtraction. During third step L time, 200_8 will be added to the value of n-m in the AC.

The MQ is checked for zero by taking the contents of the MQ to the SR zero check circuits. If MQ = 0, the column 9 carry trigger is left on, assuming it had been turned on because AC = 0, and the instruction is terminated. If MQ = 0, the column 9 carry trigger is turned off (if it was on) and the instruction proceeds.

If AC = MQ = 0, or the divide check trigger is on (FP trigger off), the instruction is terminated at the end of the first L time. With termination the following takes place:

1. Initiate End Op.
2. Turn on T2 Trigger.
3. If divide check, set AC and MQ to original dividend.
4. If AC = MQ = 0, set AC and MQ to +0. (AC reset by ADQ-35 to AC).

L Time Second Step

The objectives of second step L time include: (See single precision flow chart).

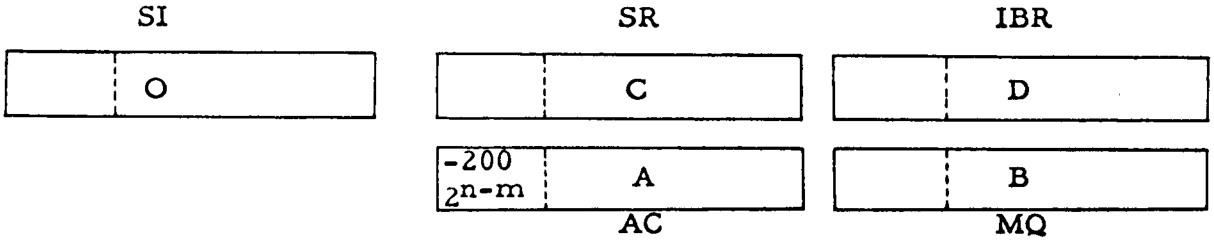
1. Increase the characteristic d:fference by one if the dividend fraction is equal to or greater than th: divisor fraction.

2. Shift the dividend fraction left one place if the dividend fraction is less than the divisor fraction (quotient will be less than 1).

These objectives are accomplished in the following manner:

1. Complement AC Q - 35 to AD.
2. AD Q - 35 to AC (the AC now holds the complement of the dividend).
3. SR 9 - 35 to AD and AC Q - 35 to AD. This step is the actual comparison of the divisor and dividend.
4. Carry to AD 8. This one to AD 8 will increment the characteristic by one. If there is no column 9 carry, the incremented contents of AD Q - 8 are returned to the AC.
5. If there is a column 9 carry the dividend is less than the divisor and the quotient will be less than one. Since the comparison is a valid reduction, a column 9 carry also calls for shifting the AC and MQ left one place and putting a one in AC 35 if MQ 9 is a 0.

Upon completion of second step L time the registers contain:



L Time Third Step

The objectives of L time third step are to:

1. Perform the division of $\frac{A + B}{C}$
2. Compute the quotient characteristic.
3. Recompute the original dividend characteristic and set into AC. (Could be + 1 - see flow chart).

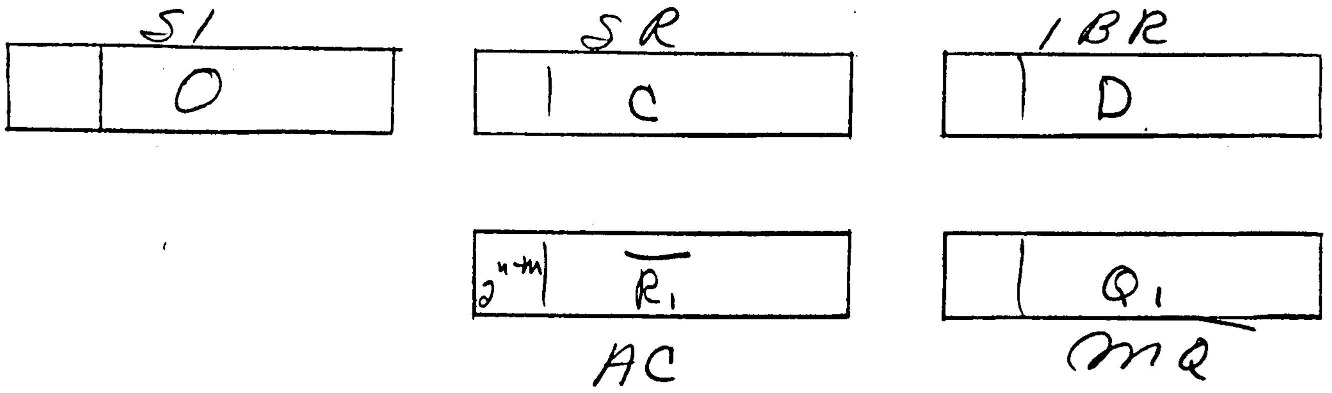
To accomplish the above objectives the computer performs a single precision floating divide. For details of this operation refer to the flow chart on single precision floating divide preceding this section.

At the completion of L time third step the computer has performed the first divide operation. The next step is to perform the first multiply function. At the end of this division the registers contain: (See page 409.7A).

MULTIPLY $Q_1 \cdot D$

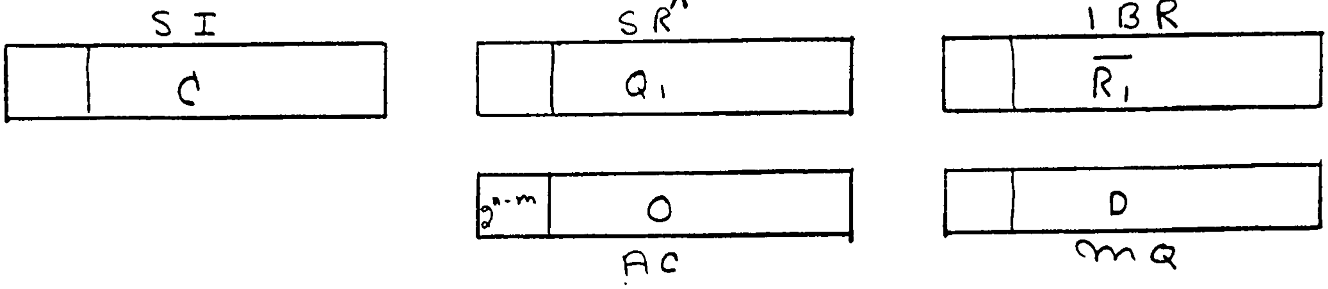
The flow chart indicates the register swapping required before the multiply step of a DPFD instruction takes place. (See page 409.13). The purpose of the multiplication

Register contents at end of first divide:

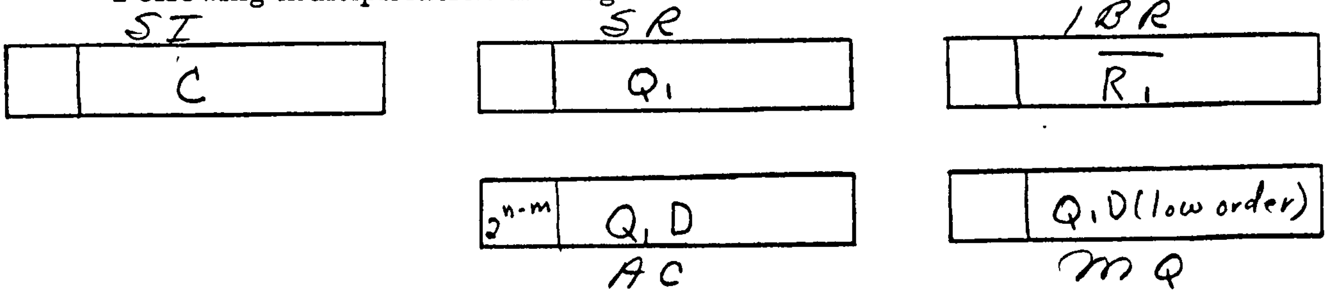


is to develop the product Q_1D for use in a subsequent subtraction $(R_1 - Q_1D)$.

At the end of the register swapping, (prior to MPY) the registers contain :

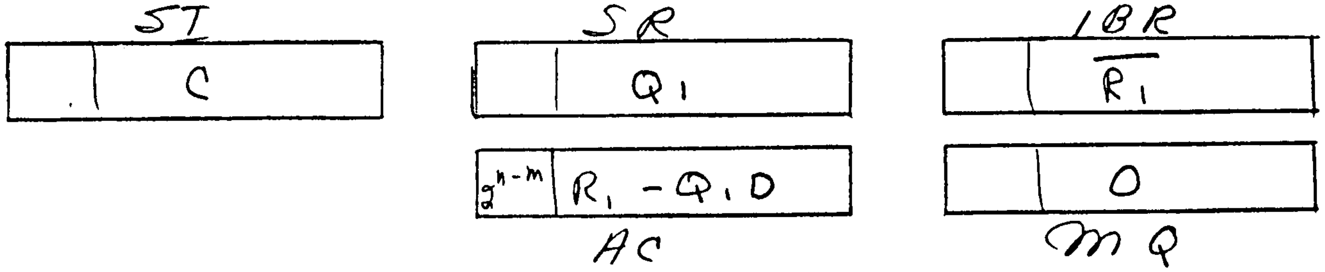


Following multiplication the registers contain :



ADDITION OF $R_1 - Q_1D$

Prior to performing this addition the AC contains the product Q_1D and the other operand, $\overline{R_1}$, is in the IBR. The IBR is gated to the SR and the addition performed. Upon completion of the addition the registers contain:

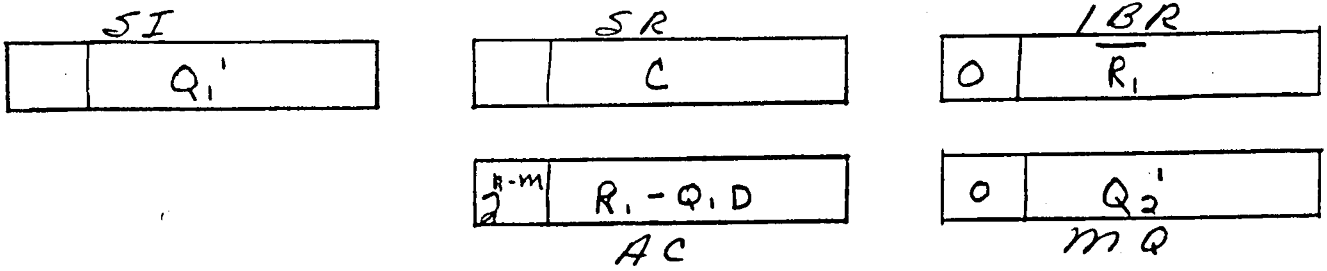


see sign determination chart on p 409.14

With addition complete the computer will perform the second and final division -- $\frac{R_1 - Q_1D}{C}$.

SECOND DIVIDE -- $\frac{R_1 - Q_1 D}{C}$

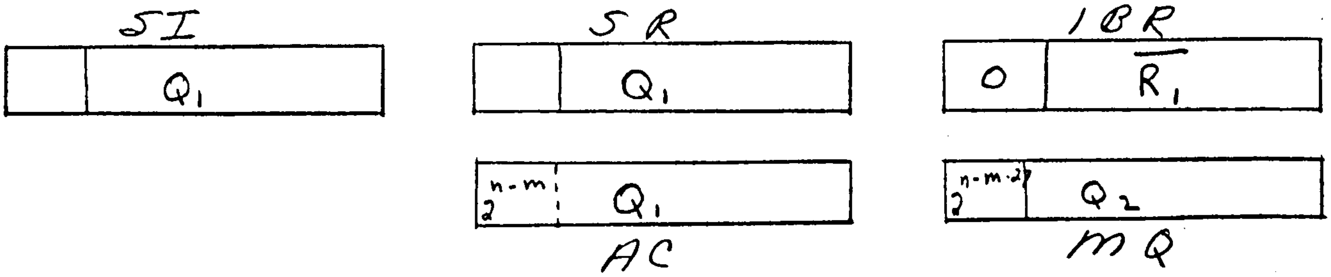
The computer performs a single precision floating point divide to find the quotient Q_2. For details of this operation refer to the single precision floating divide flow chart preceding this section. At the end of this division the registers contain :



The computer is now ready for the final addition of $Q_1 + Q_2$, which is the final answer in double precision fboating point form.

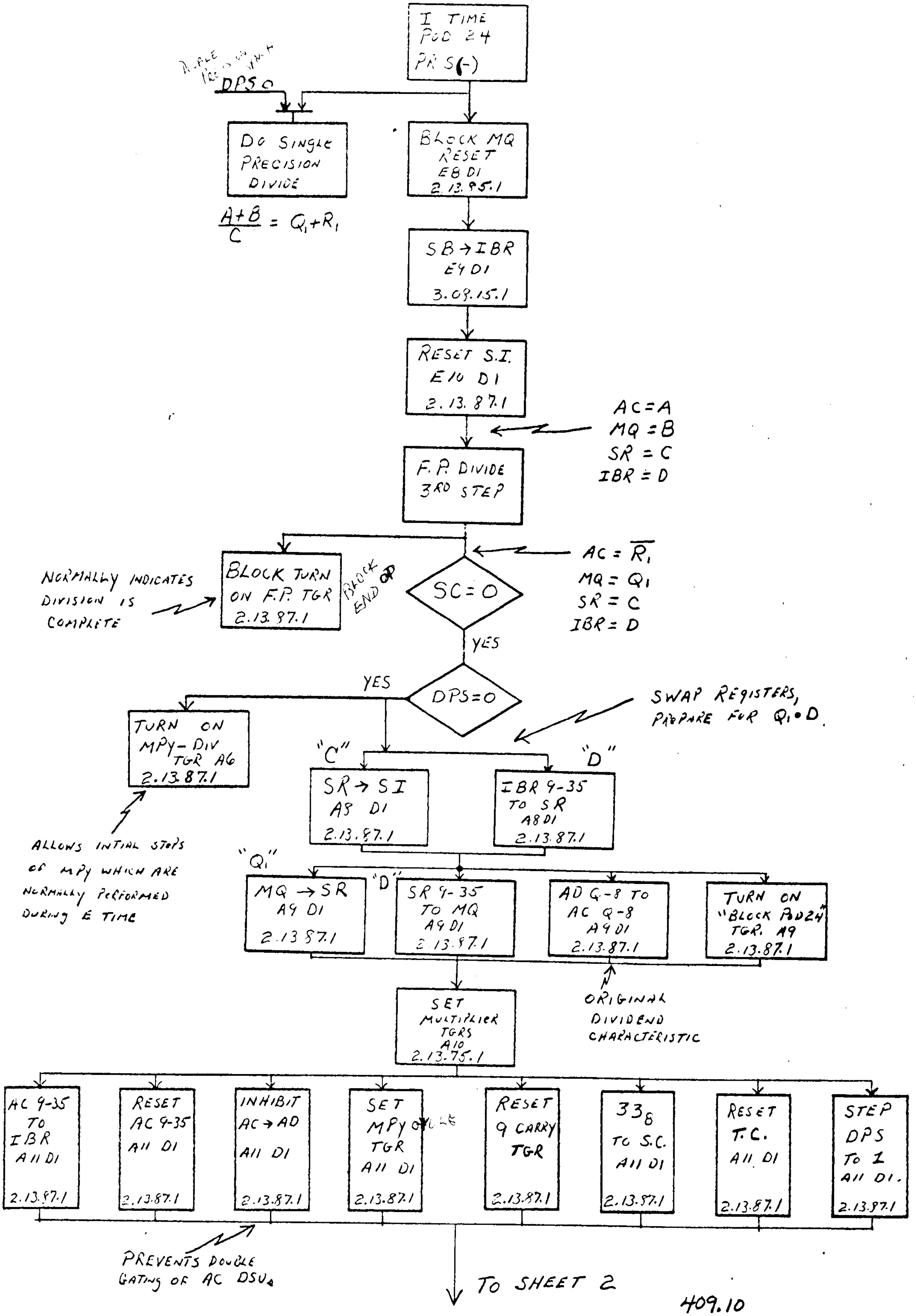
FINAL ANSWER -- ADD $Q_1 + Q_2$

In order to perform this addtion, Q_1 is gated from the IBR to the SR and the FACT 2 trigger is turned on (see section on floating add). At the end of the addition the regüsters contian the final answer as shown :

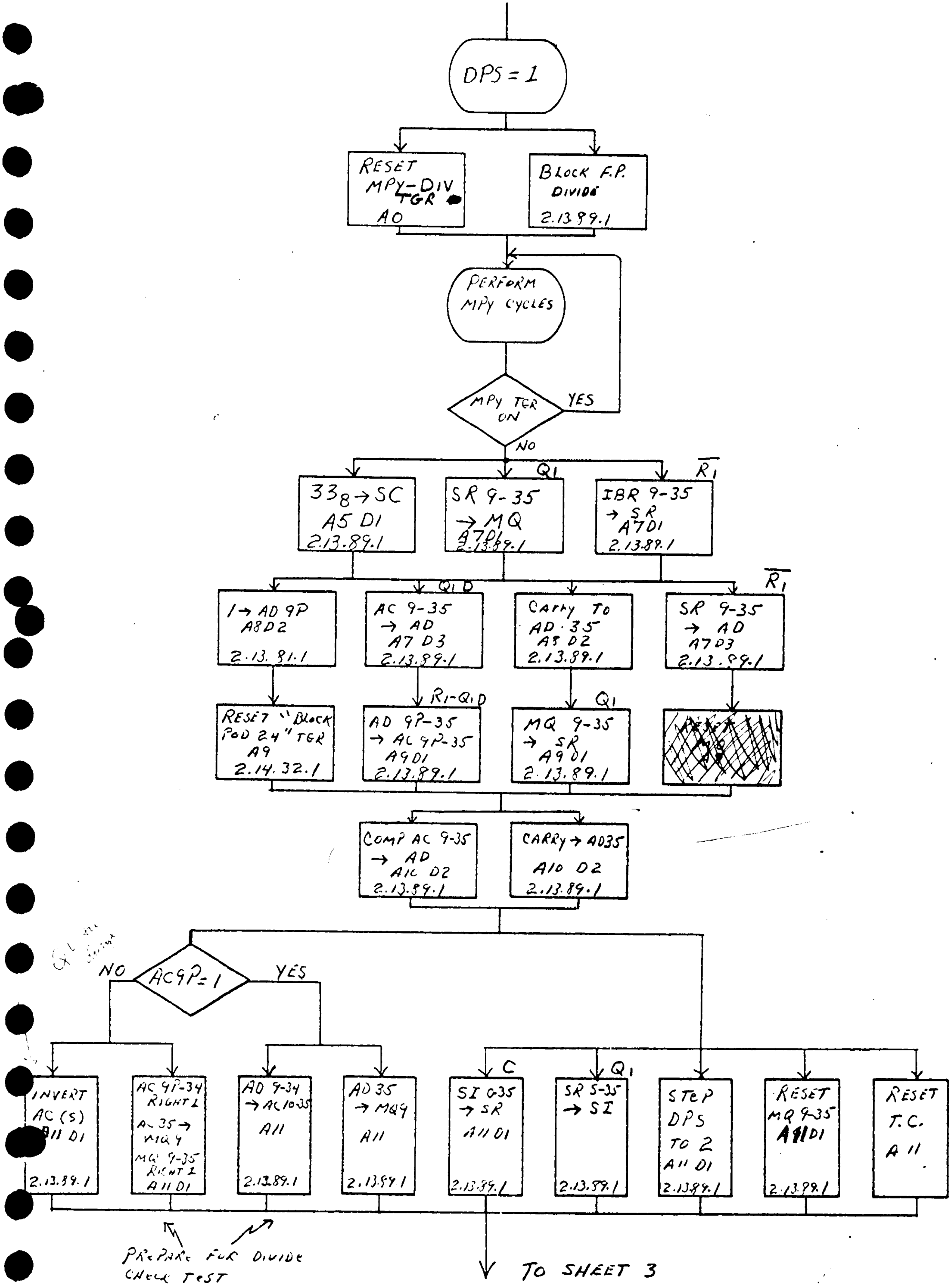


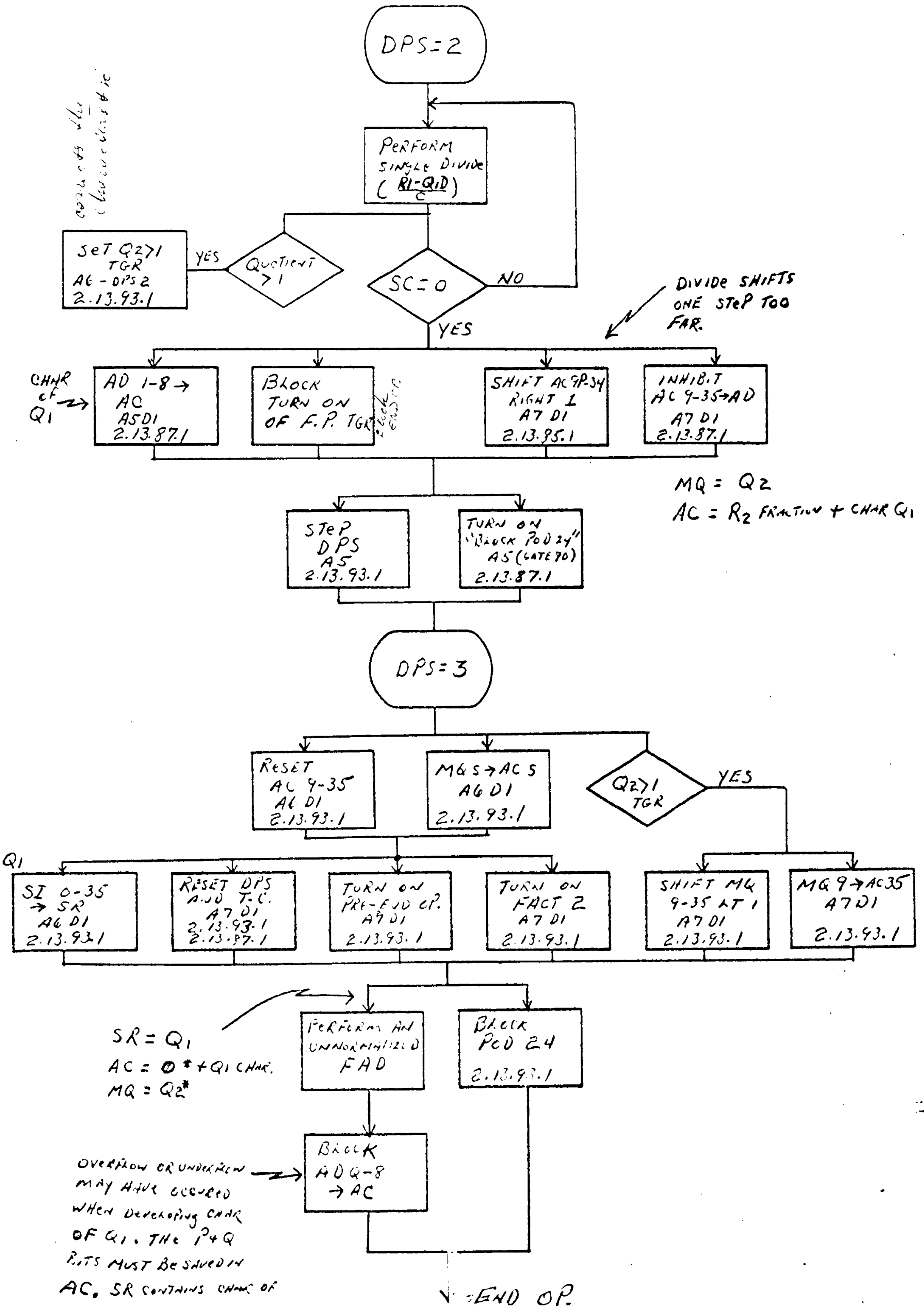
The final answer may now be stored or retained in the registersfor further arithmetic operations.

DOUBLE PRECISION FLOATING DIVIDE



DOUBLE DIVIDE


DPS = 1
RESET MPY-DIV TGR A0
BLOCK F.P. DIVIDE 2.13.89.1
PERFORM MPY CYCLES
MPY TGR ON
YES
NO
33_8 → SC A5 D1 2.13.89.1
Q1
SR 9-35 → MQ A7D1 2.13.89.1
$\overline{R_1}$
IBR 9-35 → SR A7D1 2.13.89.1
1 → AD 9P A8D2 2.13.81.1
Q1 D
AC 9-35 → AD A7 D3 2.13.89.1
CARRY TO AD·35 A8 D2 2.13.89.1
$\overline{R_1}$
SR 9-35 → AD A7D3 2.13.89.1
RESET "BLOCK POD 24" TGR A9 2.14.32.1
R1-Q1D
AD 9P-35 → AC 9P-35 A9D1 2.13.89.1
Q1
MQ 9-35 → SR A9D1 2.13.89.1
COMP AC 9-35 → AD A10 D2 2.13.89.1
CARRY → AD35 A10 D2 2.13.89.1
AC9P=1
NO
YES
INVERT AC (S) A11 D1 2.13.89.1
AC 9P-34 RIGHT 1 AC 35 → MQ 9 MQ 9-35 RIGHT 1 A11 D1
AD 9-34 → AC10-35 A11 2.13.89.1
AD 35 → MQ9 A11 2.13.89.1
C
SI 0-35 → SR A11 D1 2.13.89.1
Q1
SR S-35 → SI 2.13.89.1
STEP DPS TO 2 A11 D1 2.13.89.1
RESET MQ 9-35 A11D1 2.13.99.1
RESET T.C. A11
PREPARE FOR DIVIDE CHECK TEST
TO SHEET 3


DPS=2
PERFORM SINGLE DIVIDE (RI-QID / C)
QUOTIENT >1
YES
SET Q2>1 TGR A6 - DPS 2 2.13.93.1
SC=0
NO
YES
DIVIDE SHIFTS ONE STEP TOO FAR.
CHAR of Q1
AD 1-8 → AC A5 D1 2.13.87.1
BLOCK TURN ON OF F.P. TGR
SHIFT AC 9P-34 RIGHT 1 A7 D1 2.13.95.1
INHIBIT AC 9-35→AD A7 D1 2.13.87.1
MQ = Q2
AC = R2 FRACTION + CHAR Q1
STEP DPS A5 2.13.93.1
TURN ON "BLOCK POD 24" A5 (GATE 90) 2.13.87.1
DPS=3
RESET AC 9-35 A6 D1 2.13.93.1
MQ S→AC S A6 D1 2.13.93.1
Q2>1 TGR
YES
Q1
SI 0-35 → SR A6 D1 2.13.93.1
RESET DPS AND T.C. A7 D1 2.13.93.1 2.13.87.1
TURN ON PRE-END OP. A7 D1 2.13.93.1
TURN ON FACT 2 A7 D1 2.13.93.1
SHIFT MQ 9-35 LT 1 A7 D1 2.13.93.1
MQ 9→AC35 A7 D1 2.13.93.1
SR = Q1
AC = 0* + Q1 CHAR.
MQ = Q2*
PERFORM AN UNNORMALIZED FAD
BLOCK POD 24 2.13.93.1
OVERFLOW OR UNDERFLOW MAY HAVE OCCURED WHEN DEVELOPING CHAR OF Q1. THE P+Q BITS MUST BE SAVED IN AC. SR CONTAINS CHAR OF
BLOCK AD Q-8 → AC
END OP.

DOUBLE PRECISION DIVIDE REGISTER SHIFTING

	AC			MQ			SR			SI			IBR		
	S	Q-8	9-35	S	1-8	9-35	S	1-8	9-35	S	1-8	9-35	S	1-8	9-35
(EG D2) E7	A	A	A	B	B	B	C	C	C						
E9	A	A	A	B	B	B	C	C	C				D	D	D
E10	A	A	A	B	B	B	C	C	C	0	0	0	D	D	D
END 1ST DIV	A	A'	$\overline{R_1}$	Q_1	Q_1	Q_1'	C	C	C	0	0	0	D	D	D
L8	A	A'	$\overline{R_1}$	Q_1	Q_1	Q_1'	C	C	D	C	C	C	D	D	D
L9	A	A'	$\overline{R_1}$	Q_1	Q_1	D	Q_1	Q_1	Q_1'	C	C	C	D	D	D
L11	A	A'	0	Q_1	Q_1	D	Q_1	Q_1	Q_1'	C	C	C	0	0	$\overline{R_1}$
END MPY	A	A'	$Q_1'D$	Q_1	Q_1		Q_1	Q_1	Q_1'	C	C	C	0	0	$\overline{R_1}$
L7	A	A'	$Q_1'D$	Q_1	Q_1	Q_1'	Q_1	Q_1	$\overline{R_1}$	C	C	C	0	0	$\overline{R_1}$
L9	A	A'	$Q_1D-\overline{R_1}$	0	0	0	Q_1	Q_1	Q_1'	C	C	C	0	0	$\overline{R_1}$
L11	R_1-Q_1D	A'	R_1-Q_1D	0	0	0	C	C	C	Q_1	Q_1	Q_1'	0	0	$\overline{R_1}$
END 2ND DIV	R_1-Q_1D	Q_1	R_1-Q_1D	Q_2'	0	Q_2'	C	C	C	Q_1	Q_1	Q_1'	0	0	$\overline{R_1}$
A6	Q_2'	Q_1	0	Q_2'	0	Q_2'	Q_1	Q_1	Q_1'	Q_1	Q_1	Q_1'	0	0	$\overline{R_1}$
A7	Q_2'	Q_1	0	Q_2'	0	Q_2'	Q_1	Q_1	Q_1'	Q_1	Q_1	Q_1'	0	0	$\overline{R_1}$
FAD'	Q_1	Q_1	Q_1	Q_2	Q_2	Q_2	Q_1	Q_1	Q_1'	Q_1	Q_1	Q_1'	0	0	$\overline{R_1}$

Note: the prime(') mark indicates slight modification may have taken place e.g., +1 or -1.

R.K. 6-16-62

$A+B$	$C+D$	Q_1	R_1*	$Q_1 D$	$R_1 - Q_1 D$		Q_2	
					$R_1 > Q_1 D$	$R_1 < Q_1 D$	$R_1 > Q_1 D$	$R_1 < Q_1 D$
+	+	+	+	+	+	−	+	−
+	−	−	+	+	+	−	−	+
−	+	−	−	−	−	+	−	+
−	−	+	−	−	−	+	+	−

Sign Determination − Double Precision Floating Divide

*Note that R_1 follows the sign ŋ $A+B$

500 ~~5.3.00~~ INSTRUCTIONS

501 ~~5.3.01~~ Word Transmission Instructions*

Word transmission instructions are necessary to move information into and out of the [illegible] ~~CPU~~. Factors must be brought in from core storage and results must be returned. Instructions are available to move parts of words or whole words to or from core storage.

Store STO +0601(I, E) Figure 501-1 ~~5.3-1~~ 5LA

This instruction moves a full word to core storage. The contents of the AC(S, 1-35) replace the contents of storage location X. The AC is unchanged. The store prefix, decrement, tag, address, and MF store control lines are activated so a full word can be put into core storage.

Store Logical Word SLW +0602(I, E) Figure 501-1 ~~5.3-1~~ 5LA

This instruction replaces the contents of storage location X with the contents of the AC(P-35). The AC is unchanged. Execution of this instruction is identical to that of store except for the routing of AC(P) to the storage register sign position. AC(P-35) is gated to SR(S-35) on Systems 2.12.02.1.

Store MQ STQ -0600(I, E) Figure 501-1 ~~5.3-1~~ 5LA

The contents of core storage location X are replaced by the contents of the MQ(S, 1-35) register. The contents of the MQ are unchanged. This instruction operates similarly to store, except that the word sent to the SR comes from the MQ rather than from the AC.

Store Zero STZ +0600(I, E) Figure 501-1 ~~5.3-1~~ 5LA

This instruction causes zeros to be placed in all positions of storage location X. The operation of this instruction is similar to store, except that nothing is put on the SB. Therefore, when the storage bus is gated to core storage, zeros are put into that location of storage.

Store Prefix STP +0630(I, E) Figure 501-2 ~~5.3-2~~ 5LA

This instruction places the contents of AC positions (P, 1 and 2) into core storage location X, positions (S, 1 and 2). The contents of the AC and storage positions (3 through 35) are unchanged. MF store prefix and MF store control are activated to cause core storage to store what is on SB(S, 1 and 2). To get the information to the storage bus, it must be moved from the accumulator to the SR.

~~*See Store Location and Trap, and Figure 5.3-34~~.

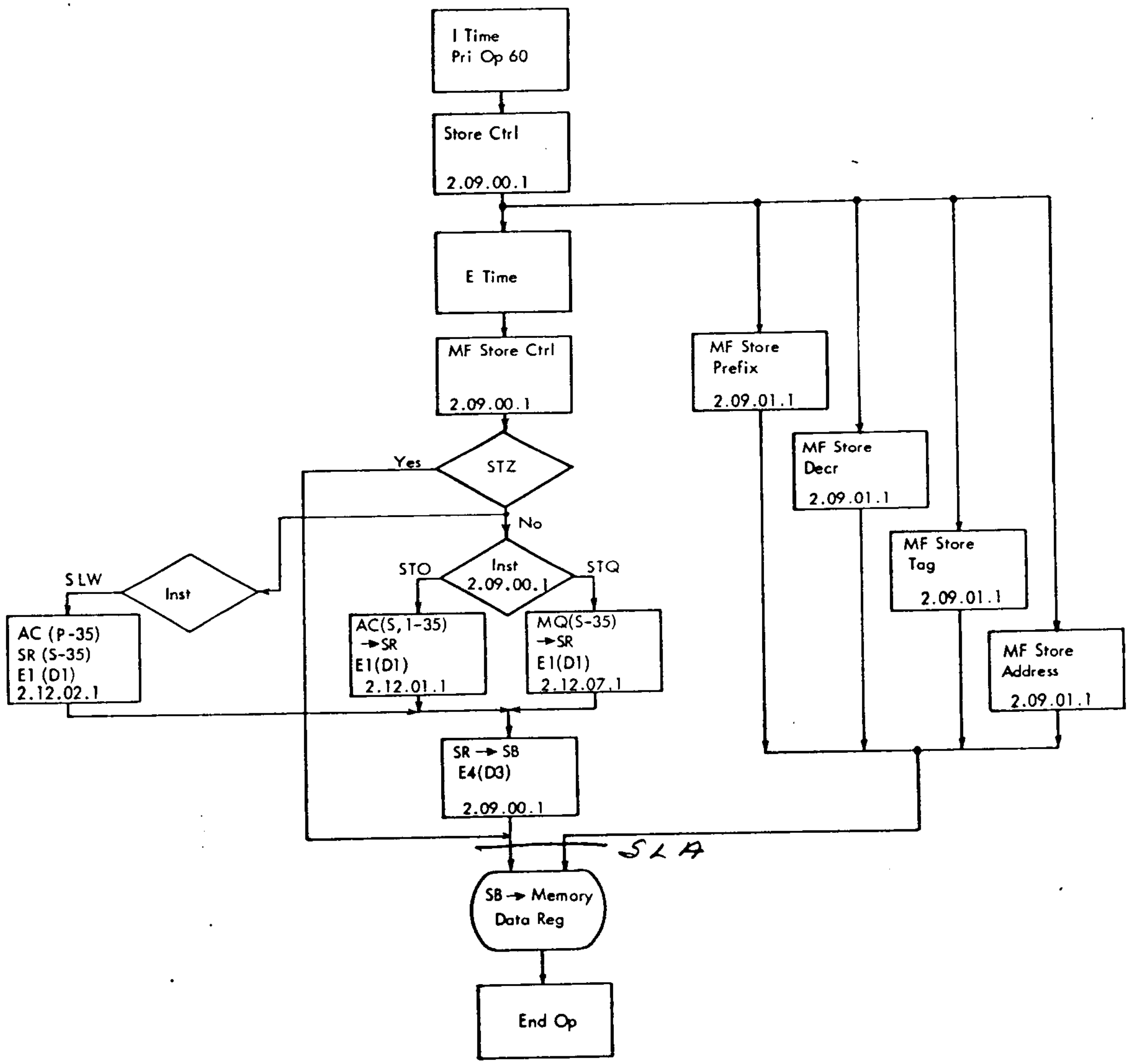


FIGURE ~~5.3-1~~ 501-1. STO + 0601; STQ - 0600; STZ + 0600; SLW + 602

Store Decrement STD +0622(I,E) SLA Figure ~~5.3-2~~ 501-2

The contents of AC(3-17) replace the contents of positions (3-17) of core storage location X. The remaining storage positions are unchanged, and the AC is unchanged. This instruction operates similarly to store prefix, except that MF store decrement rather than MF store prefix is made active.

Store Tag STT +0625(I,E) SLA Figure ~~5.3-2~~ 501-2

The contents of AC(18-20) replace the contents of positions (18-20) of core storage location X. The AC and the remaining positions of core storage are unchanged. This instruction operates similarly to store prefix, except that MF store tag rather than MF store prefix is made active.

Store Address STA +0621(I,E) SLA Figure ~~5.3-2~~ 501-2

The contents of AC(21-35) replace the contents of positions (21-35) of core storage location X. The remaining positions of storage and the contents of the AC are unchanged. This instruction also operates similarly to store prefix, except that MF store address rather than MF store prefix is made active.

Store Left Half MQ SLQ -0620(I,E) SLA Figure ~~5.3-3~~ 501-3

The contents of positions (S-17) of the MQ replace the contents of positions (S-17) of storage location X. The remaining storage positions and the contents of the MQ are unchanged. MF store prefix, MF store decrement, and MF store control lines are activated so SB(S-17) can be read into storage. The word is sent from the MQ to the SR so it can be put on the SB.

Store Instruction Location Counter STL -0625(I,E) SLA Figure ~~5.3-4~~ 501-4

The contents of the program counter, which contains the location of the STL instruction plus one, replace the contents of positions (21-35) of storage location X. The contents of the PC and the remaining positions of storage are unchanged. The MF store address and MF store control lines are activated so the address portion of the storage word can be changed. The PC contents are sent to core storage on the SB. The contents of the PC are sent to the SR through the ~~AS~~ XAP and the output of the SR feeds the SB.

Load MQ LDQ +0560(I,E) TLA or DLA Figure ~~5.3-5~~ 501-5

The contents of the MQ are replaced by the contents of storage location X. The storage word is put into the SR, and the output of the SR is sent to the MQ.

Exchange AC and MQ XCA +0131 (I) Figure ~~5.3-6~~ 501-6

The contents of the AC(S,1-35) are interchanged with the contents of the MQ(S,1-35). AC positions Q and P are reset. The SR is used for temporary storage while routing the MQ to the AC.

Exchange Logical Accumulator and MQ XCL -0130(I) Figure ~~5.3-7~~ 501-7

This instruction interchanges the contents of AC(P-35) and the contents of the MQ (S-35). Positions (S) and (Q) of the AC are cleared. Execution is identical to that of XCA except for the handling of AC positions (S) and (P). The contents of the MQ are put into the SR. The contents of the SR(S, 1-35) and the AC (P,1-35) are interchanged, putting the original MQ in the AC. Gating the output of the SR(S,1-35) to the MQ(S,1-35) places the original AC contents in the MQ. AC positions (Q) and (S) are cleared because of normal shift cell operations.

STL

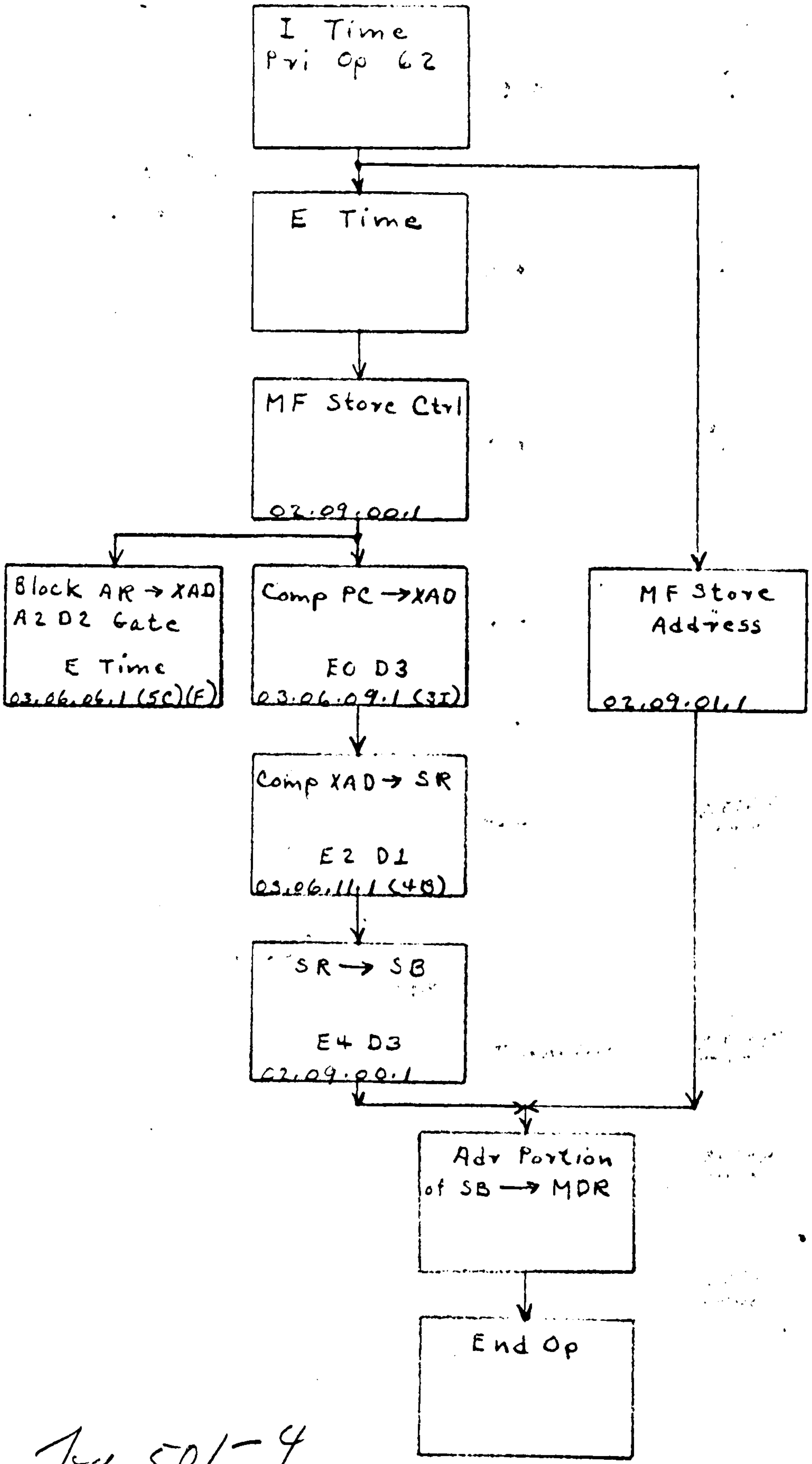
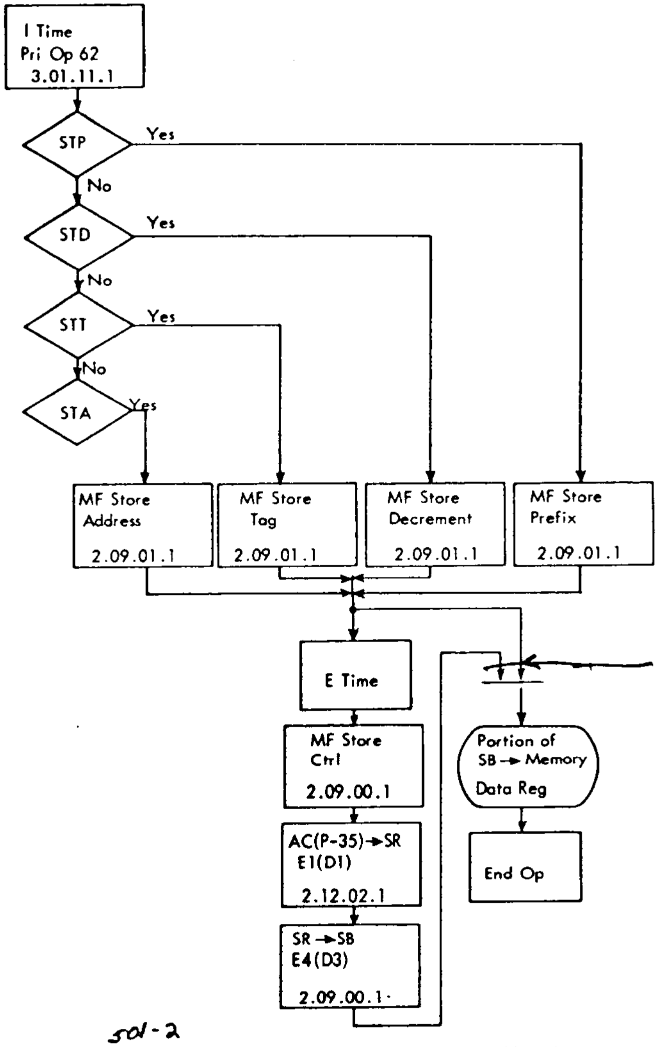


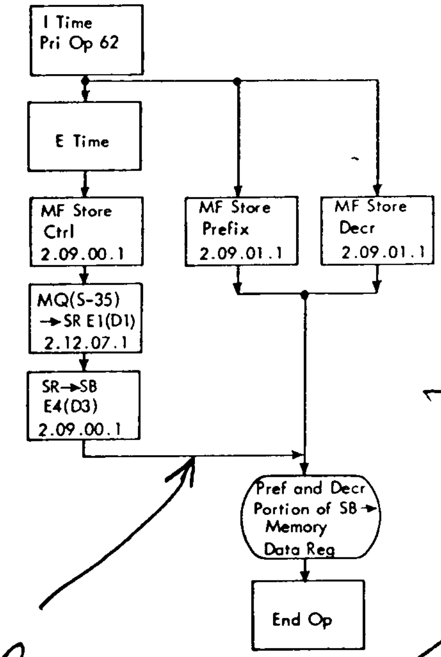
Fig 501-4
p 500.3.

11-20-61 WEK

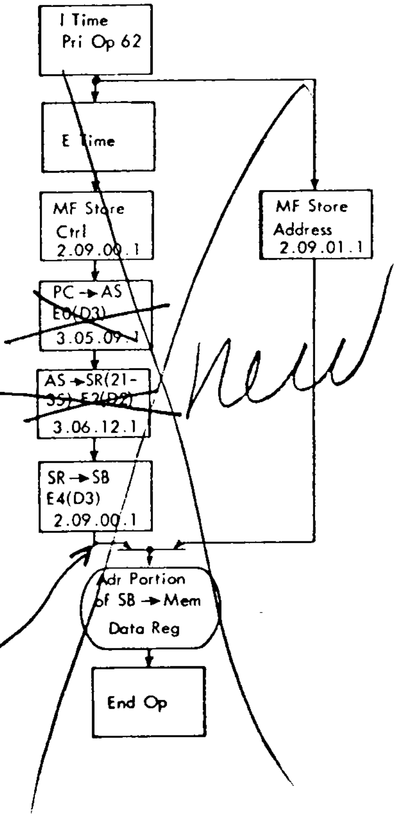


SLA
Here

501-2
FIGURE ~~5.3-2~~. STP +0630; STD +0622; STT +0625; STA +0621



501-3
FIGURE ~~5.3-3~~. SLQ - 0620



501-4
FIGURE ~~5.3-4~~. STL -0625

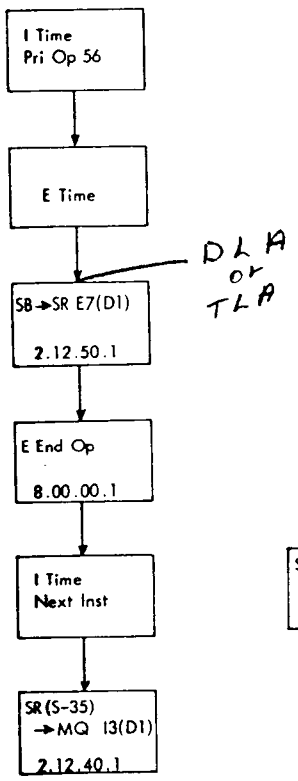


FIGURE ~~5.3-5~~ 501-5. LDQ +0560

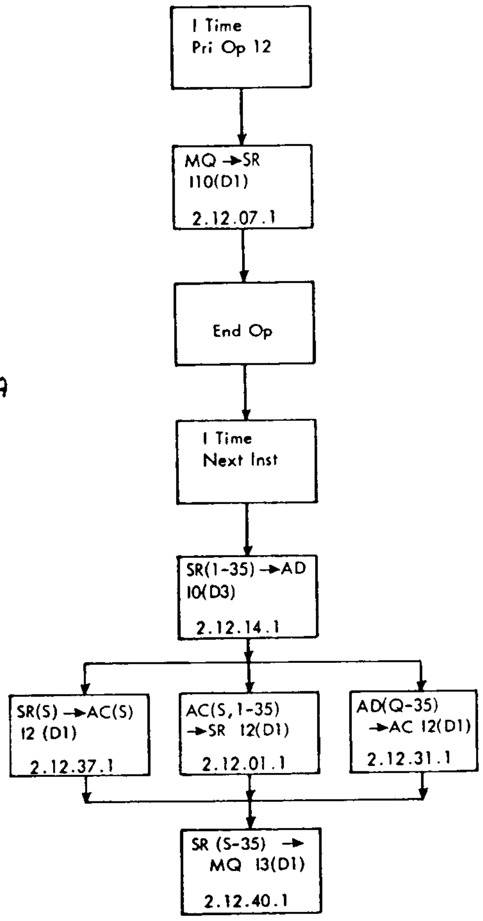


FIGURE ~~5.3-6~~ 501-6. XCA +0131

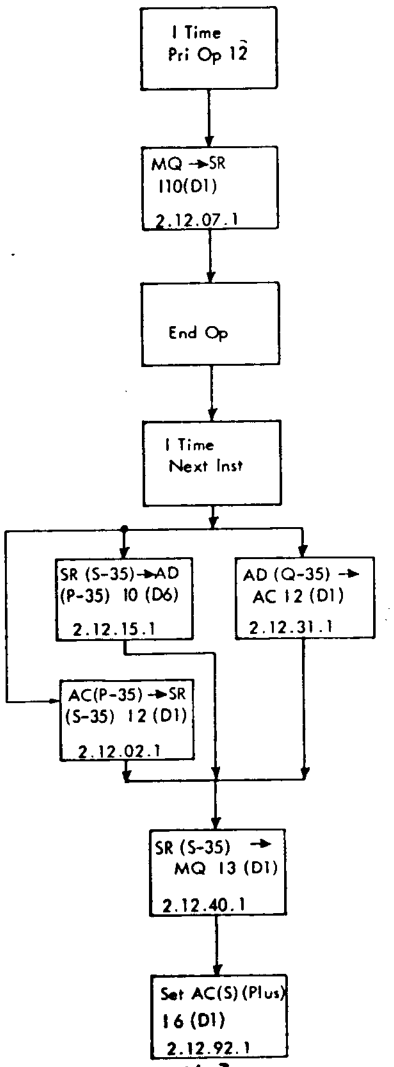


FIGURE ~~5.3-7~~ 501-7. XCL -0130

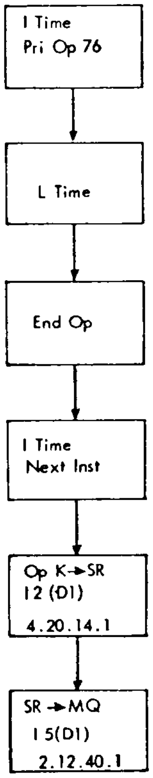
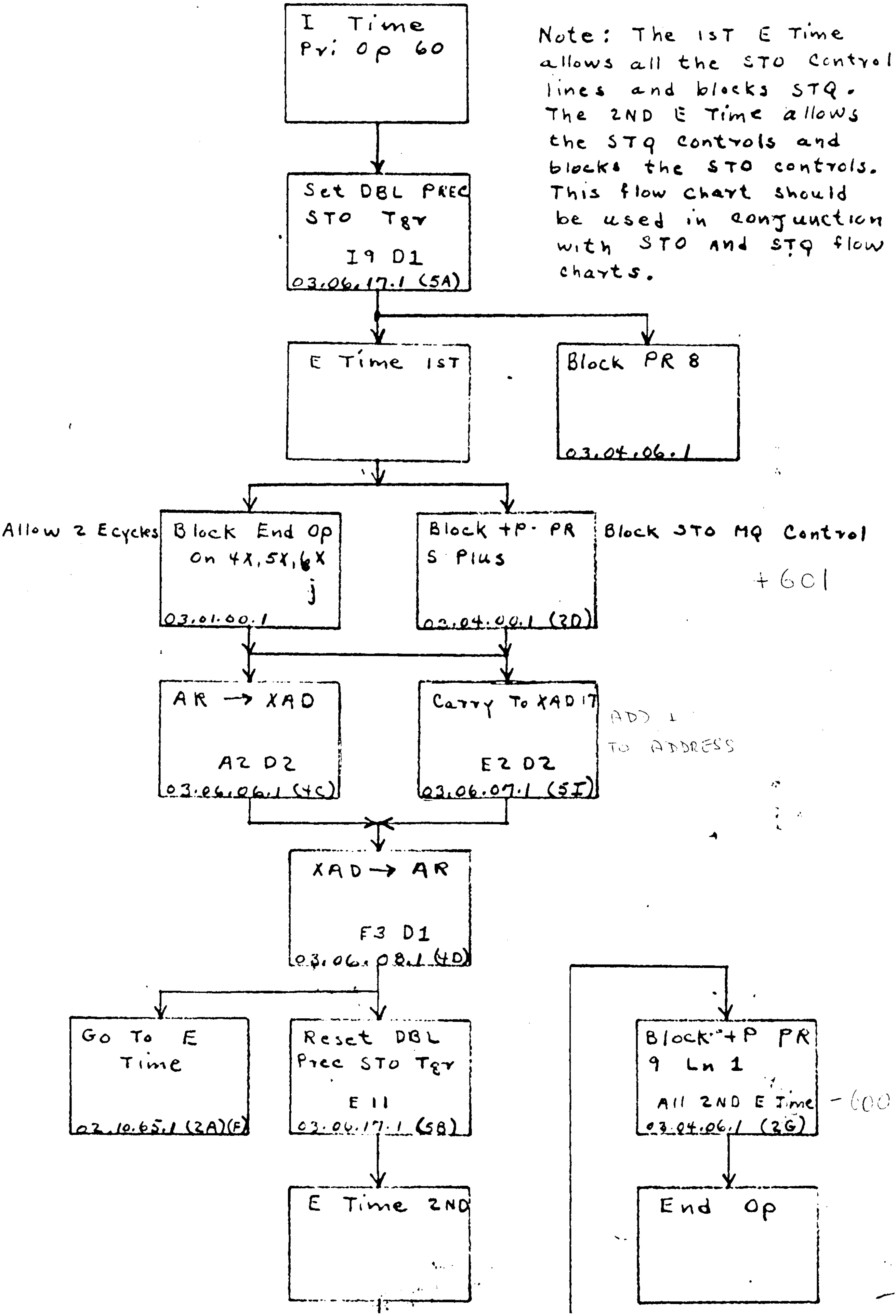
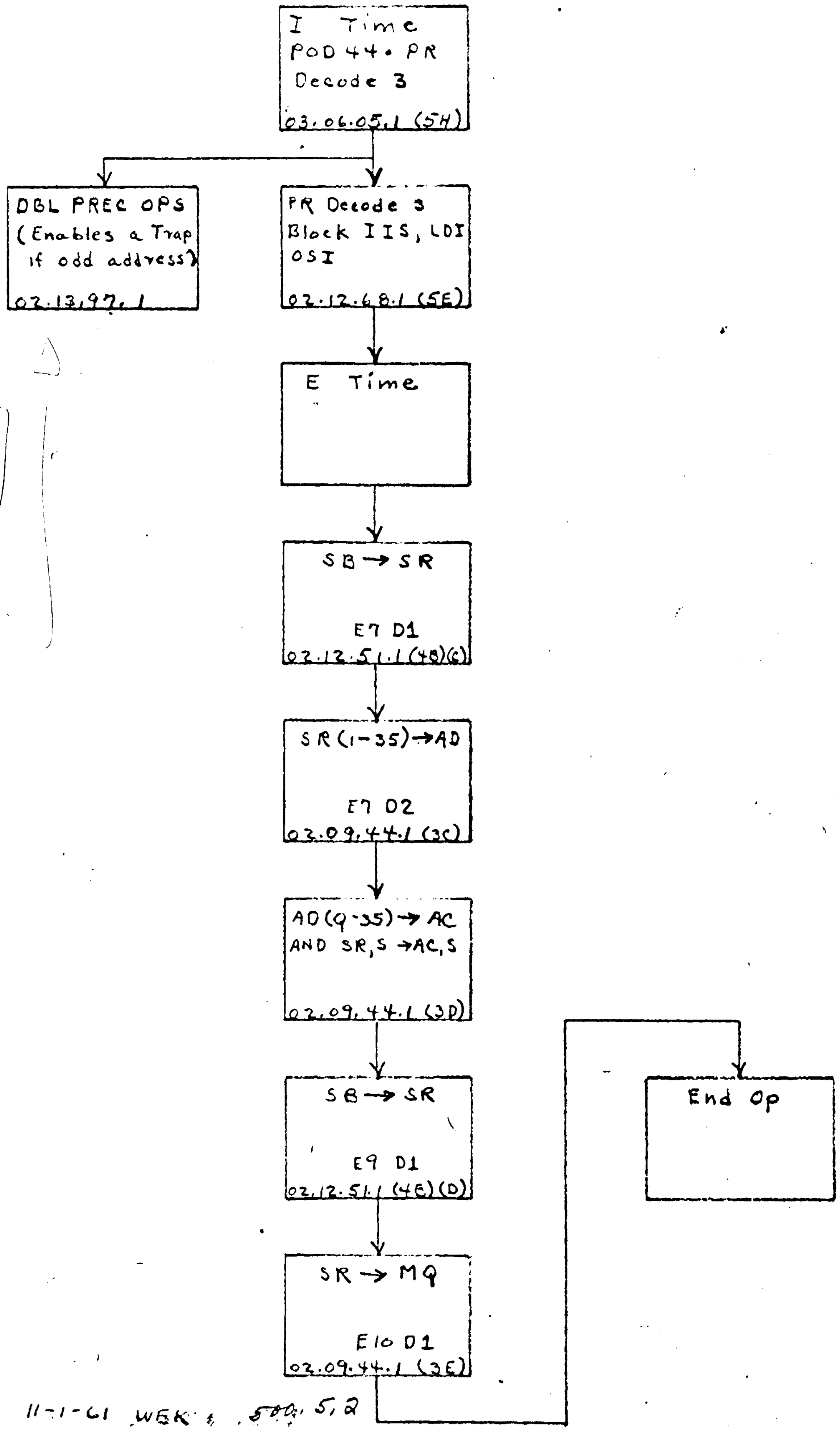


FIGURE ~~5.3-8~~ 501-8. ENK +0760...0004

Note: The 1ST E Time allows all the STO Control lines and blocks STQ. The 2ND E Time allows the STQ controls and blocks the STO controls. This flow chart should be used in conjunction with STO and STQ flow charts.



DBL LOAD +0443



SPILL BIT IS POS 12

11-1-61 WEK 500.5,2

501-8

Enter Keys   ENK +0760...0004 (I,L) DLA Figure ~~5.3-8~~

The word represented in console keys (S, 1-35) replaces the contents of the MQ. This is a primary operation 76 instruction. The word in the keys is put into the SR and the output of the SR is gated into the MQ.

502

~~5.3.02~~ Shifting Instructions

Shift instructions are used to align words, or for fast multiplication or division by a power of 2. Shifting moves the bits of the AC or MQ, or both, to the right or left within the registers. Bits shifted out of the end of a register are lost and bits shifted away from either end of a register are replaced by zeros. Because the shift counter receives only the last eight positions of the instruction address, the maximum number of shifts possible is 255_{10} ($11\ 111\ 111_2$).

The number of shifts to be taken is indicated by the address portion of the shift instruction. This address is gated to the shift counter at I11(D1). Shifting starts at the next L1 time and continues to shift at the rate of one position of shift for each clock pulse until the shift counter has been reduced to zero.

The cyclic makeup of a shifting instruction is an I time followed with as many L times as required to complete the shifts designated by the address portion of the instruction. Eleven shifts may be completed during the first L time, and twelve shifts may be completed during all succeeding L times. The "end operation" condition is signaled by having the shift counter at seven or less at any L10 time of a shift instruction. This means that up to five shifts may be made in the I time of the following instruction.

Shifting to the left is the same as multiplying by a power of 2; shifting to the right reduces or divides by a power of 2. The number of shifts is equal to the exponent.

502-1

Accumulator Left Shift   ALS +0767 (I, L...) DLA   Figure ~~5.3-9~~

This instruction causes the contents of the AC (Q-35)* to be shifted left a number of places equal to the eight low order positions of the address. Zeros replace any bits shifted away from position (35). Bits shifted past Q are lost. A "1" shifted into the P position turns on the AC overflow indicator.

502-2

Long Left Shift   LLS +0763 (I, L...)   DLA   Figure ~~5.3-10~~

For this instruction the contents of the MQ and AC (except the sign positions) are shifted left the number of places designated by the eight low order positions of the address. The MQ (1) position is shifted to the AC (35) position. The AC sign is set to agree with the MQ sign. Bits shifted past Q are lost and bits shifted away from MQ(35) are replaced by zeros. Bits shifted into P cause the AC overflow indicator to be turned on.

502-3

Logical Left Shift   LGL -0763 (I, L...)   DLA   Figure ~~5.3-10~~

This instruction shifts the contents of AC(Q-35) and MQ(S-35) left the number of places designated by the address. Bits shifted from MQ(1) enter MQ(S), and from MQ (S) enter AC(35). Bits shifted into AC(P) cause the AC overflow indicator to be turned on. Bits shifted past AC(Q) are lost and bits shifted away from MQ(35) are replaced by zeros. The operation of LGL is similar to LLS except for handling of the MQ sign.

* This text section uses (Q - 35) to represent (Q, P, 1 - 35).

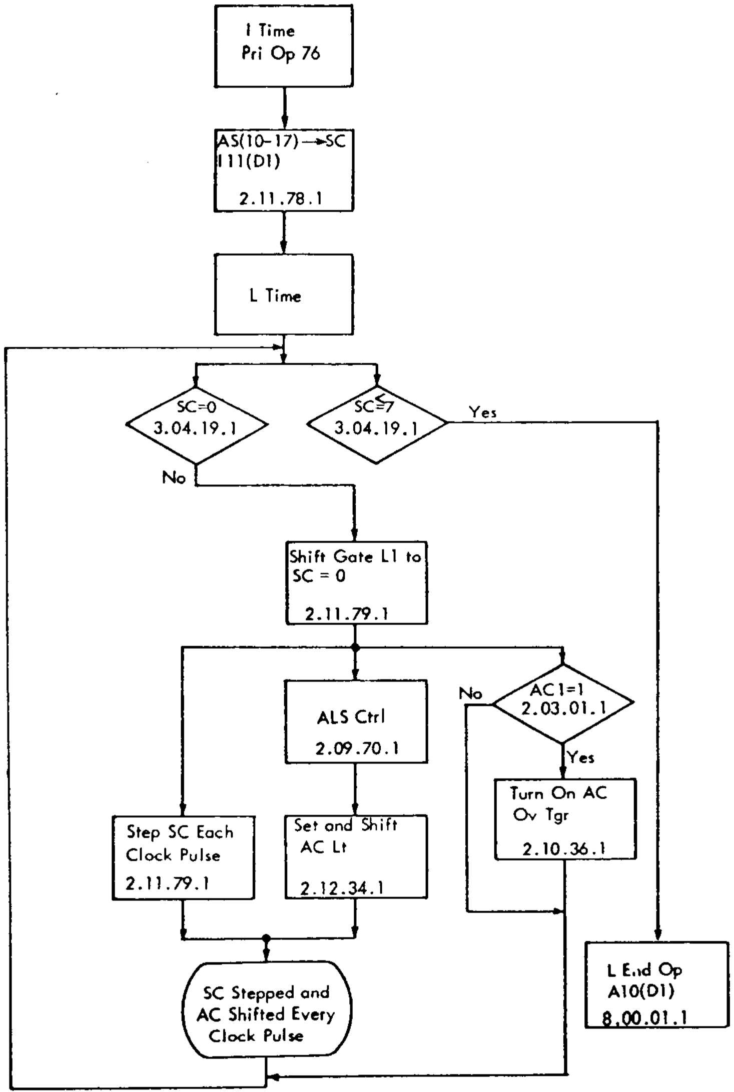


FIGURE ~~5.3-9~~. ALS +0767
502-1

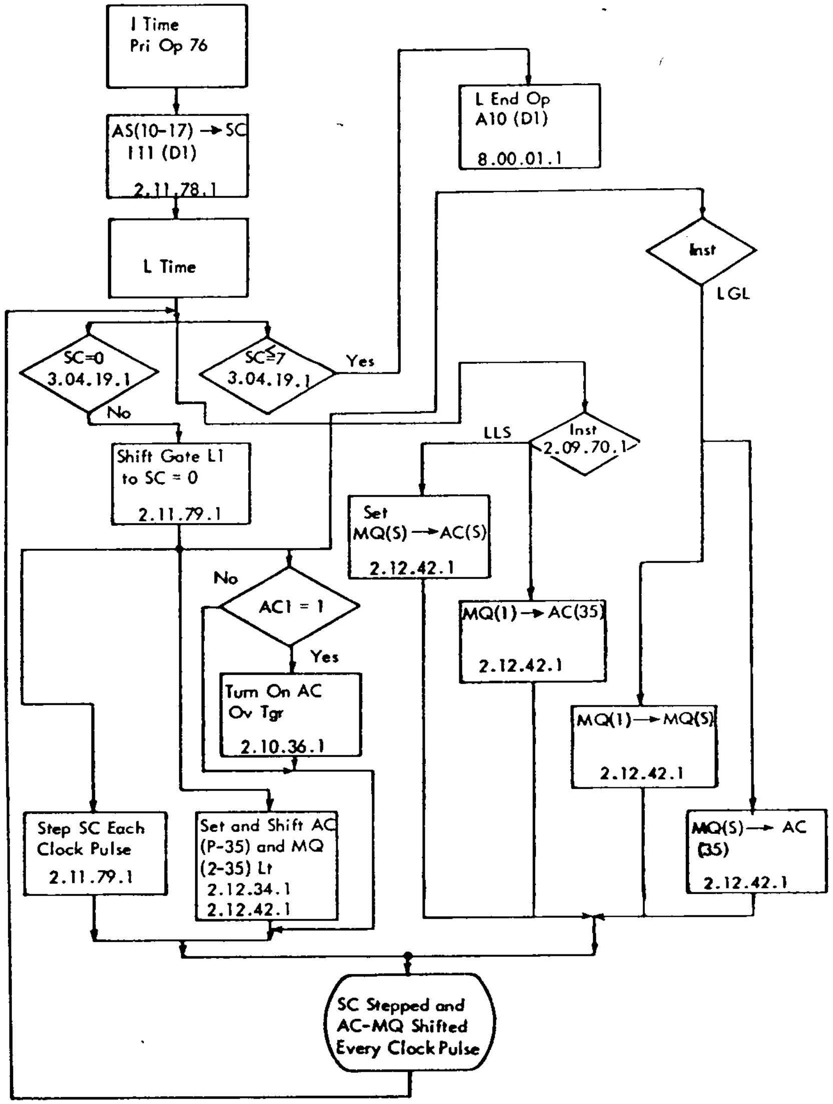


FIGURE ~~5.3-10.~~ LLS +0763; LGL -0763
502-2

Accumulator Right Shift   ARS +0771 (I, L. . .) DLA Figure ~~5.3-11~~ 502-3

This instruction causes the contents of AC (Q-35) to be shifted right the number of places indicated by the address. Bits shifted away from Q are replaced by zeros; bits shifted past position (35) are lost. ARS is similar to ALS except for direction of the shifting.

Long Right Shift   LRS +0765 (I, L. . .) DLA Figure ~~5.3-12~~ 502-4

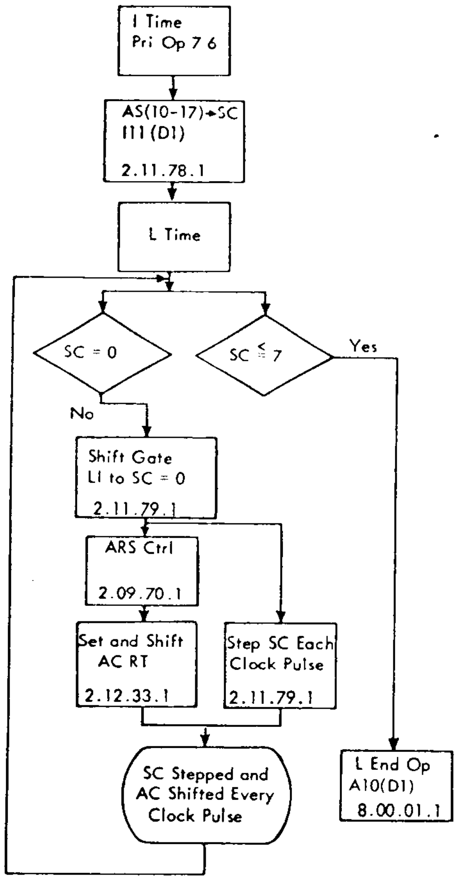
The contents of the MQ and AC (except the sign positions) are shifted right the number of places indicated by the address. The MQ sign is set to agree with the AC sign. Bits shifted away from Q are replaced by zeros and shifted past AC(35) enter MQ(1). Bits shifted past MQ(35) are lost.

Logical Right Shift   LGR -0765 (I, L. . .) DLA Figure ~~5.3-12~~ 502-4

This instruction shifts the contents of the AC(Q-35) and MQ(S-35) right the number of places designated by the address. Bits shifted out of AC(35) are entered into MQ(S) and from MQ(S) to MQ(1). Bits shifted past MQ(35) are lost. The operation of LGR is the same as LRS except for the handling of the MQ sign.

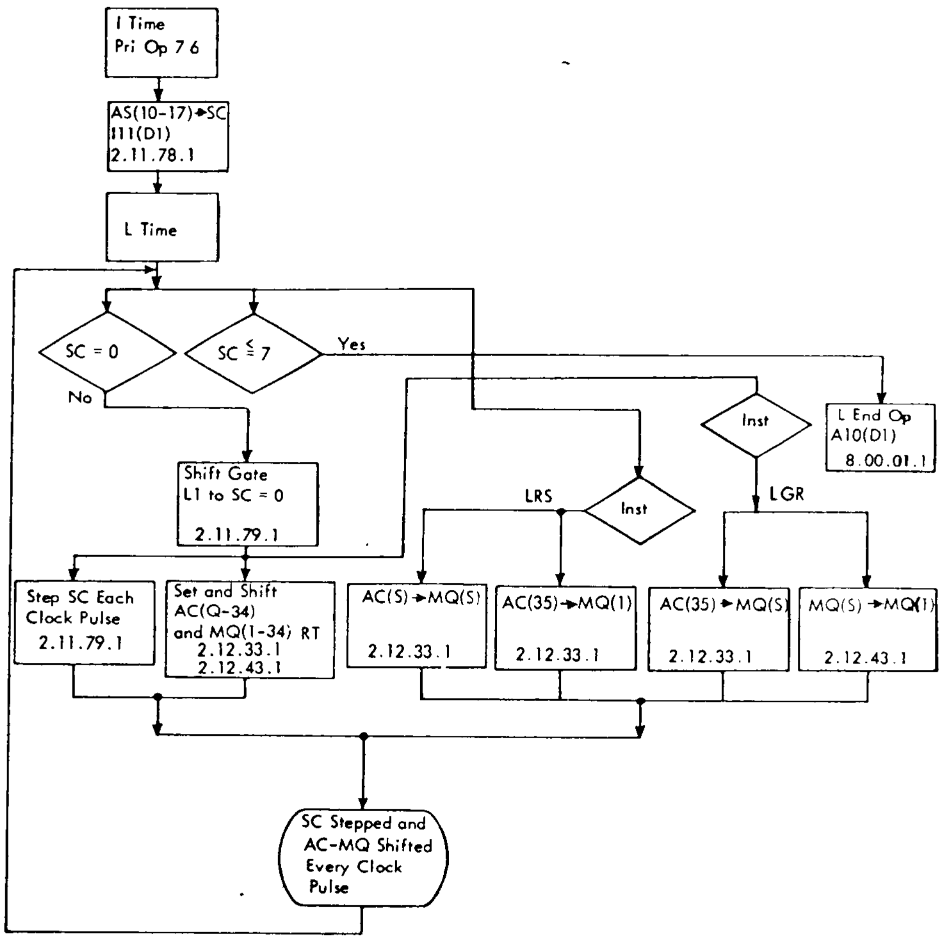
Rotate MQ Left   RQL -0773 (I, L. . .) DLA Figure ~~5.3-13~~ 502-5

This instruction shifts the contents of the MQ, including the sign, left the number of places designated by the address. Bits shifted out of MQ(1) enter the sign position and from the sign position enter MQ(35).



502-3

FIGURE ~~5.3-11~~. ARS +0771



502-4

FIGURE ~~5.3-12.~~ LRS +0765; LGR -0765

RQL - 0773

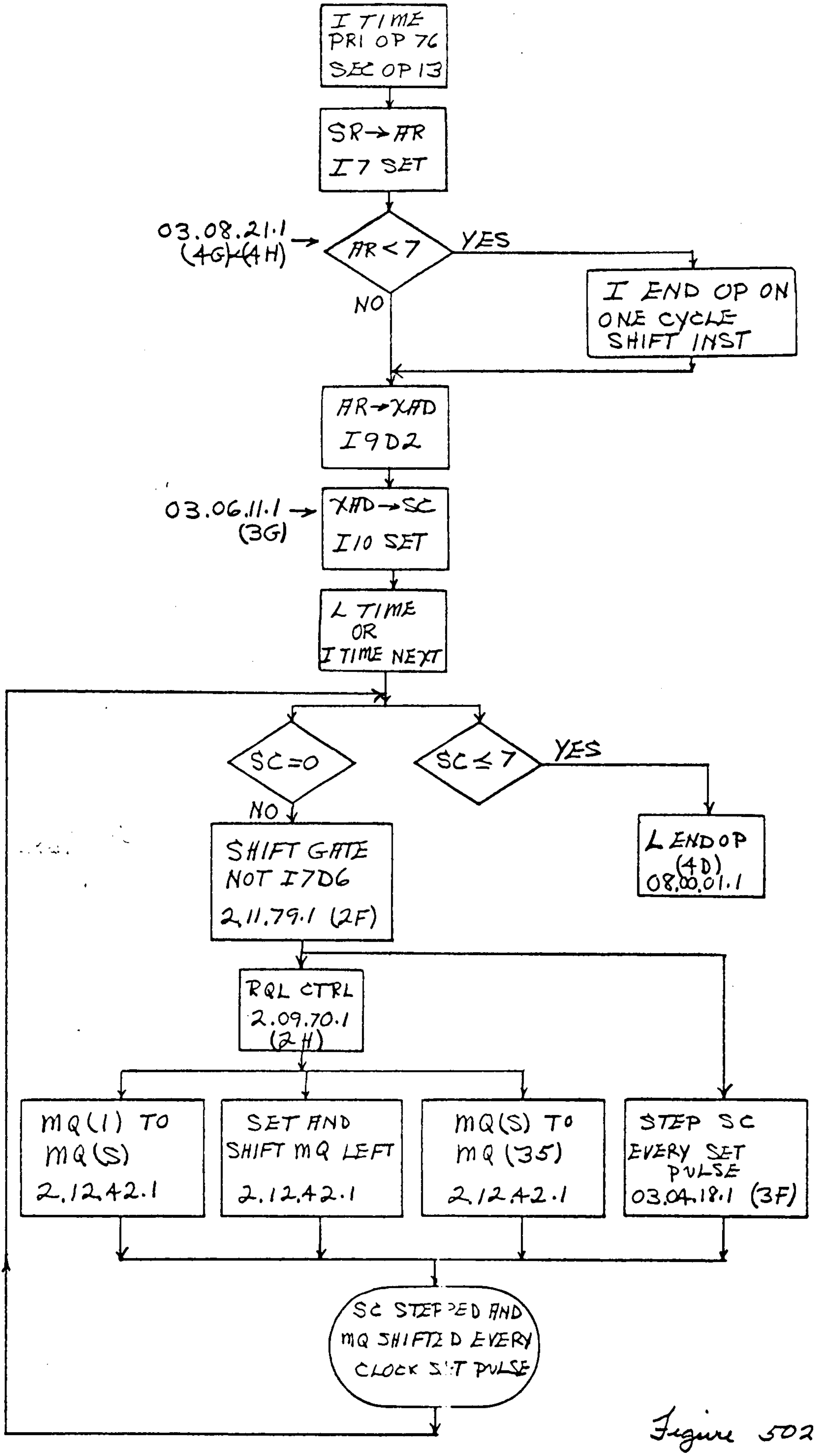


Figure 502-5

Transfer Instructions

Transfer instructions are used to alter the sequence of instructions. The conditional transfers allow automatic testing of problem conditions without stopping the computer. The transfer instructions greatly reduce program length by allowing program loops and subroutine operation

Transfer TRA +0020 (I) Figure ~~5.3-27~~ 503-1

The transfer instruction causes the computer to take its next instruction from location X and resets the instruction counter to X. The address portion of the transfer instruction is substituted for the instruction counter when setting the address register to locate the next instruction. The instruction counter is then set to this new address, and the instruction sequence continues from the new location.

Transfer on MQ Plus TQP +0162(I) Figure ~~5.3-28~~ 503-2

If the sign of the MQ register is positive, a transfer will be taken to storage location X. If the MQ sign is minus, the computer proceeds to the next instruction in sequence.

Transfer on Plus TPL +0120 (I)

If the sign of the accumulator is plus, a transfer is taken to storage location X. If the sign is minus, the computer proceeds to the next instruction in sequence. TPL is executed in the same manner as TQP, except that the AC sign rather than the MQ sign is tested. See Systems 2.10.08.1.

Transfer on Minus TMI -0120 (I)

If the sign of the AC is minus, a transfer is taken to storage location X. If the AC sign is plus, the computer proceeds to the next instruction in sequence. The execution of this instruction is the same as TQP, except that the AC sign rather than the MQ sign is tested. See Systems 2.10.08.1.

Transfer on Overflow TOV +0140 (I)

If the AC overflow trigger is on as a result of a previous operation, a transfer is taken to storage location X, and the overflow trigger is turned off. If the overflow trigger is off, the computer proceeds to the next instruction in sequence. Execution of TOV is much like TQP, except that the overflow trigger rather than the MQ sign is tested. See Systems 2.10.08.1. The AC overflow trigger is turned off on Systems 2.10.36.1.

Transfer on No Overflow TNO -0140 (I)

If the AC overflow trigger is off, the next instruction is taken from storage location X. If the overflow trigger is on, no transfer is taken, and the overflow trigger is turned off. Operation of this instruction is the same as TQP, except that the overflow trigger rather than the MQ sign is tested.

Transfer on Quotient Overflow TQO +0161 (I)

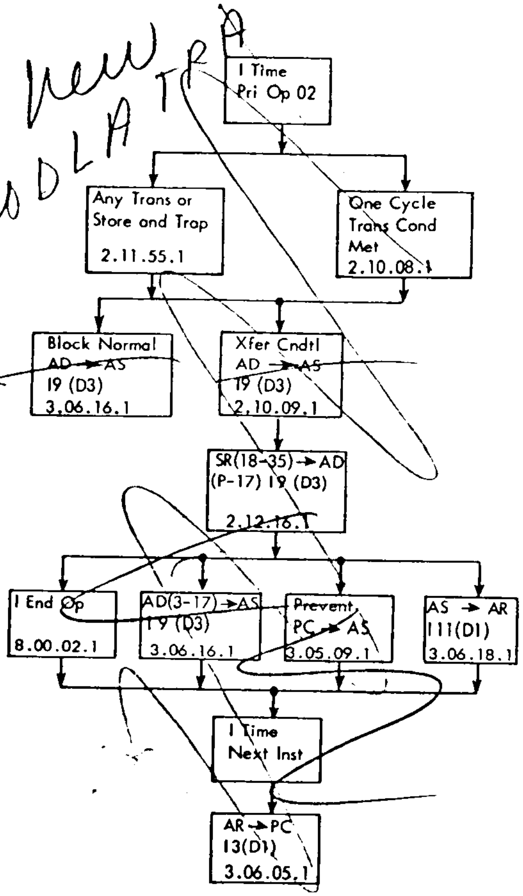
If the quotient overflow is on because of a previous operation, a transfer is taken to storage location X, and the quotient overflow trigger is turned off. If the quotient overflow trigger is off, the computer proceeds to the next instruction in sequence. This is a 704 compatibility instruction.

Transfer on Zero TZE +0100 (I, ~~E~~) Figure ~~5.3-29~~ 503-3

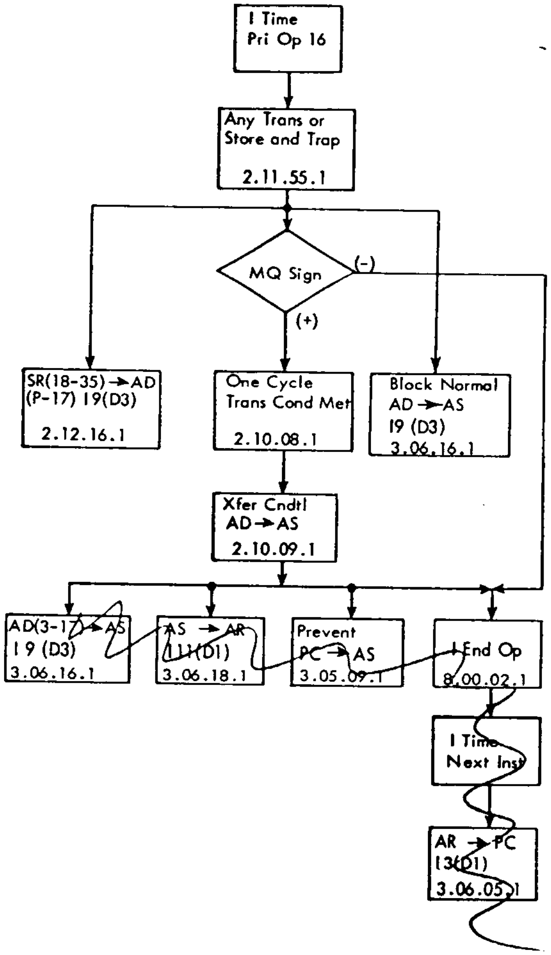
If the contents of the AC (including the overflow positions) are zero, a transfer is taken to storage location X. If the contents are not zero, the computer proceeds to the next instruction in sequence. In either case the contents of the AC are not changed.

~~85~~
500.12

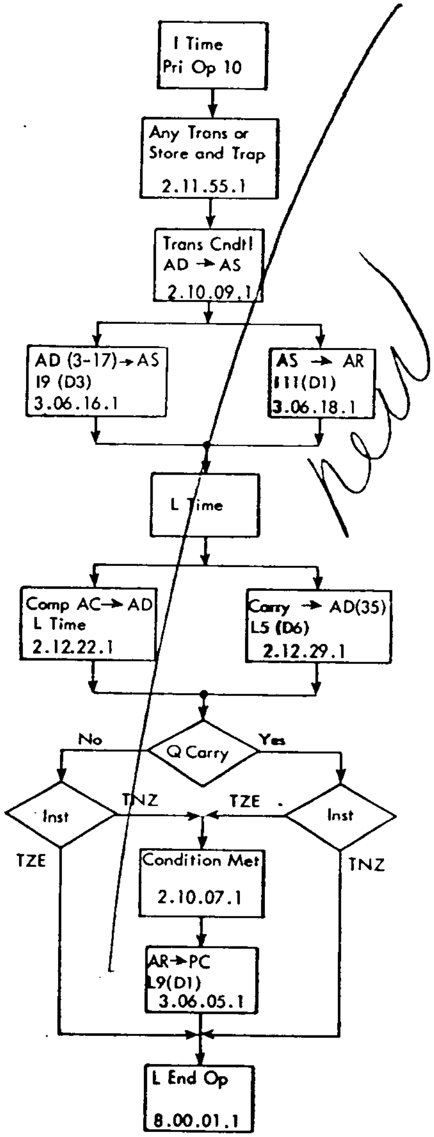
new
also DLA TRA



503-1
FIGURE ~~5.3-27~~. TRA + 0020

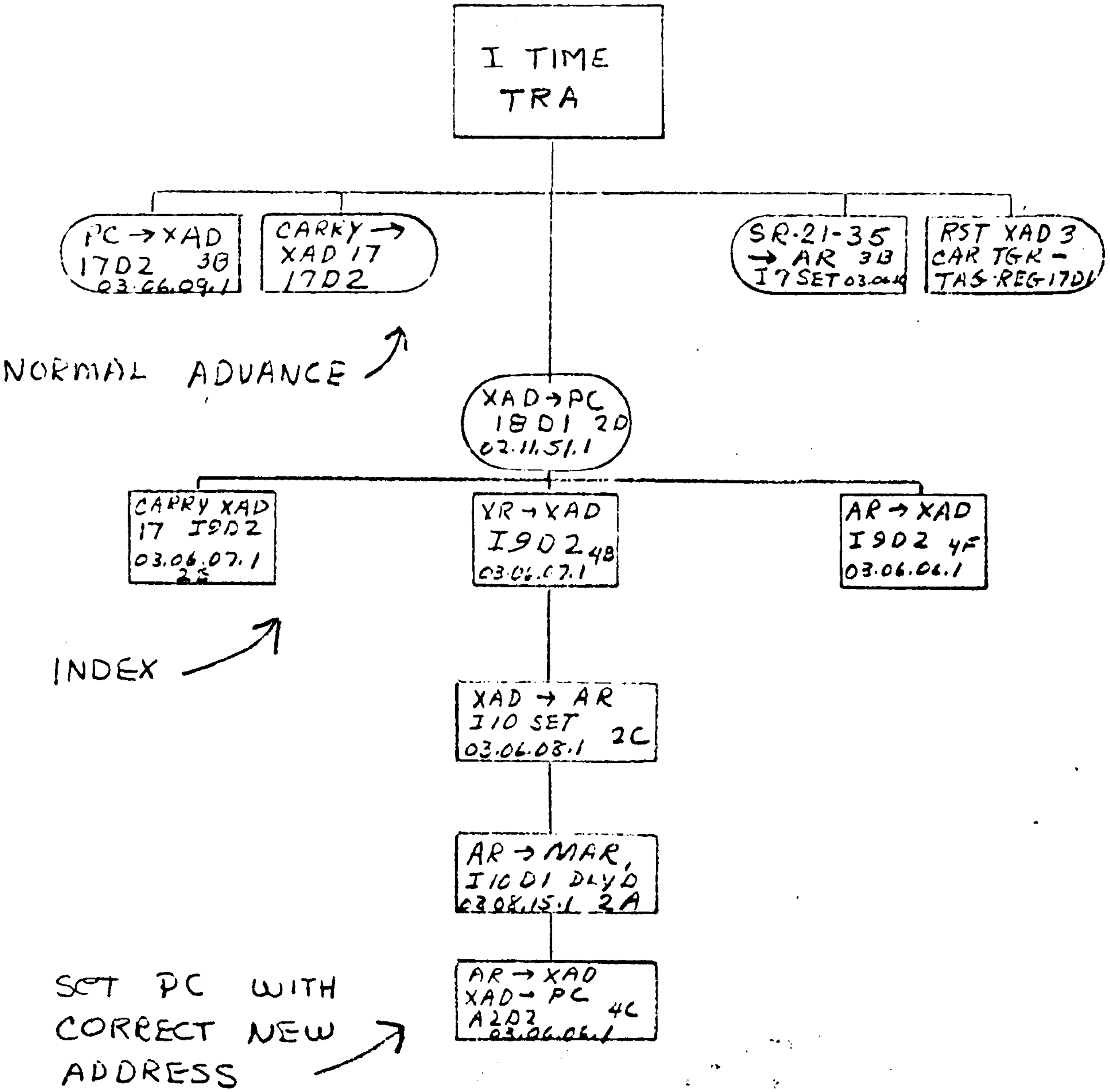


503-2
FIGURE ~~5.3-28~~. TQP + 0162



new

FIGURE ~~5.3-29~~. TZE + ~~0100;~~ TNZ - 0100
503-3


I TIME
TRA
PC → XAD
17D2 3B
03.06.09.1
CARKY →
XAD 17
17D2
SR-21-35
→ AR 3B
I7 SET 03.06.10
RST XAD 3
CAR TGR –
TAS REG 17D1
NORMAL ADVANCE
XAD → PC
18D1 2D
02.11.51.1
CARRY XAD
17 I9D2
03.06.07.1
2E
YR → XAD
I9D2 4B
03.06.07.1
AR → XAD
I9D2 4F
03.06.06.1
INDEX
XAD → AR
I10 SET 2C
03.06.08.1
AR → MAR,
I10D1 DLY'D
03.08.15.1 2A
AR → XAD
XAD → PC
A2D2 4C
03.06.06.1
SET PC WITH
CORRECT NEW
ADDRESS

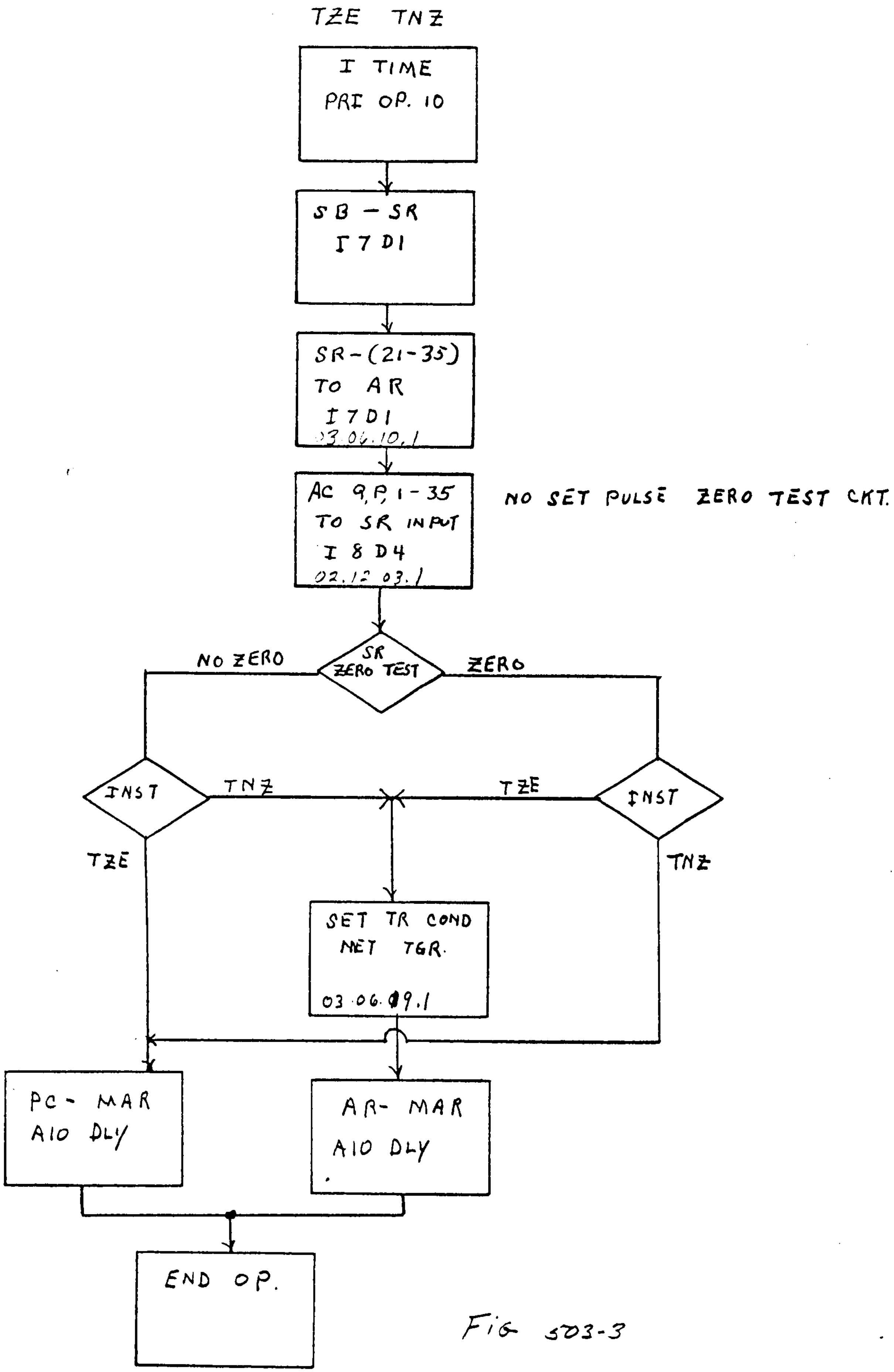

TZE TNZ
I TIME
PRI OP. 10
SB - SR
I 7 D1
SR-(21-35)
TO AR
I 7 D1
03.06.10.1
AC 9, P, 1-35
TO SR INPUT
I 8 D4
02.12 03.1
NO SET PULSE ZERO TEST CKT.
SR
ZERO TEST
NO ZERO
ZERO
INST
TNZ
TZE
INST
TZE
TNZ
SET TR COND
MET TGR.
03 06 09.1
PC - MAR
A10 DLY
AR - MAR
A10 DLY
END OP.

Fig 503-3

To test for a zero condition, the accumulator is complemented to the adders, and a one is added to position 35. If a zero condition exists, a carry results which ripples through all of the adders and turns on the Q carry trigger. The Q carry trigger is used to condition the transfer circuits.

Transfer on No Zero TNZ -0100 (I, L) Figure ~~5.3-29~~ 502-3

If the contents of the AC are not zero, the next instruction is taken from storage location X. If the contents are zero, the computer proceeds to the next instruction in sequence.

Transfer on Low MQ TLQ +0040 (I, L) Figure ~~5.3-30~~ 502-4

An algebraic comparison is made between the MQ and the accumulator contents. If the MQ contents are less than the accumulator contents, an instruction transfer is made to storage location X. If the MQ is greater or equal, no transfer is taken. For this instruction a +0 is considered to be larger than a -0. The contents of both registers are left unchanged.

The operation is performed by adding the contents of the MQ to the complemented accumulator contents. The Q carry trigger is then matched with the register signs to determine whether the conditions for transfer have been met. To prevent transfers when the factors are equal, a one is added to AC(35) to produce a Q carry when the AC factor is plus.

The following table illustrates comparisons which might be made, along with the desired result:

No. in AC	No. in MQ	Q Carry	Transfer
-7	-6	No	No
-6	-6	No	No
-0	-6	Yes	Yes
-0	-0	No	No
+0	-0	Yes	Yes
-0	+0	No	No
-6	+6	No	No
-0	+6	Yes	No
+6	+6	Yes	No
+7	+6	No	Yes

Transfer on Channel in Operation TCO +XXXX (I, L) Figure ~~5.3-31~~ 502-5

This instruction causes a transfer to storage location X if the particular data channel is in use. Operation codes +0060 through +0067 are used to check data channels A through H, respectively.

Transfer on Channel Not in Operation TCN -XXXX (I, L) Figure ~~5.3-31~~ 502-5

TCN causes a transfer to storage location X if the data channel is not in operation. Operation codes -0060 through -0067 are used to select data channels A through H, respectively.

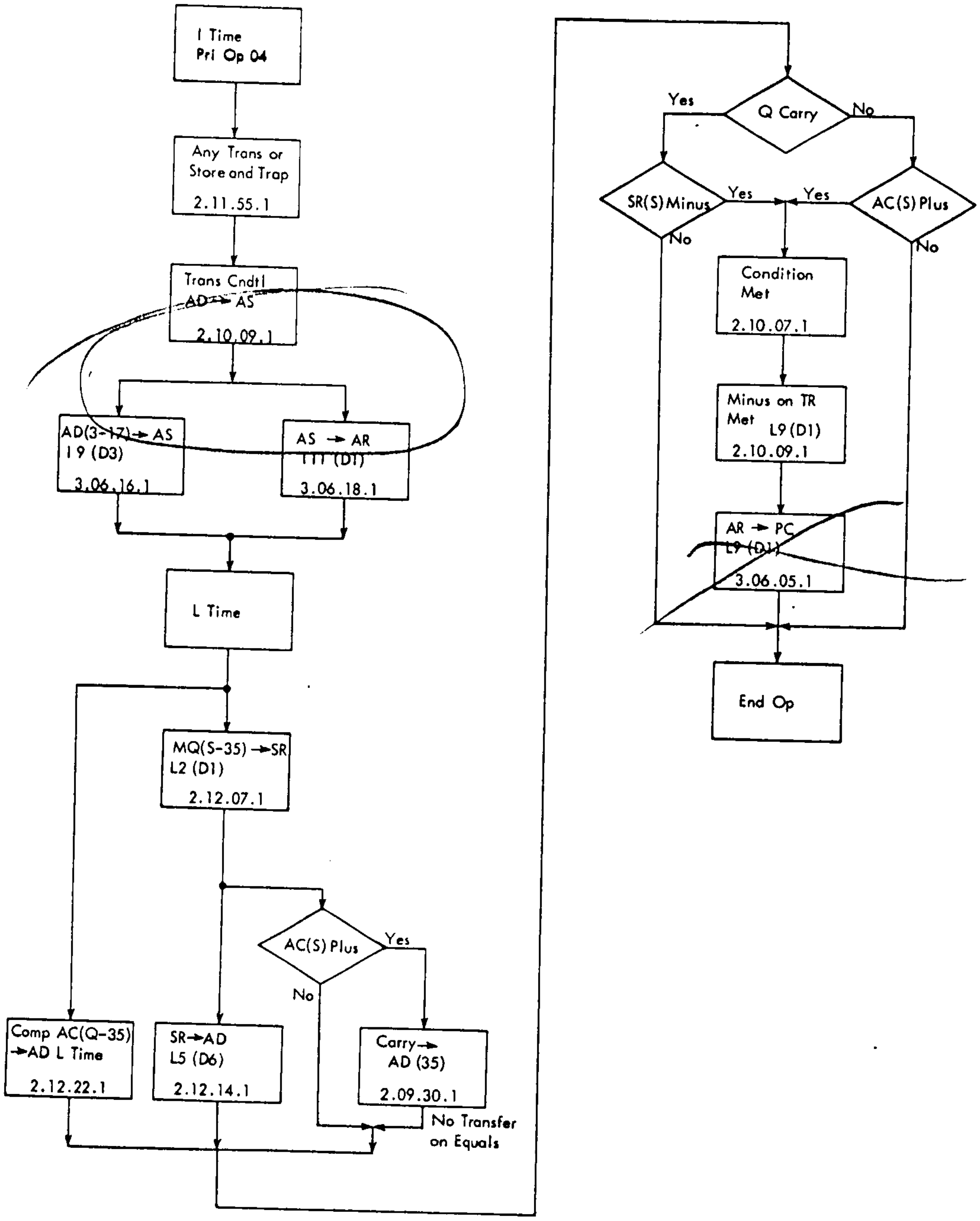


FIGURE ~~5-3-30~~ 502-4. TLQ + 0040

TCO, TCN

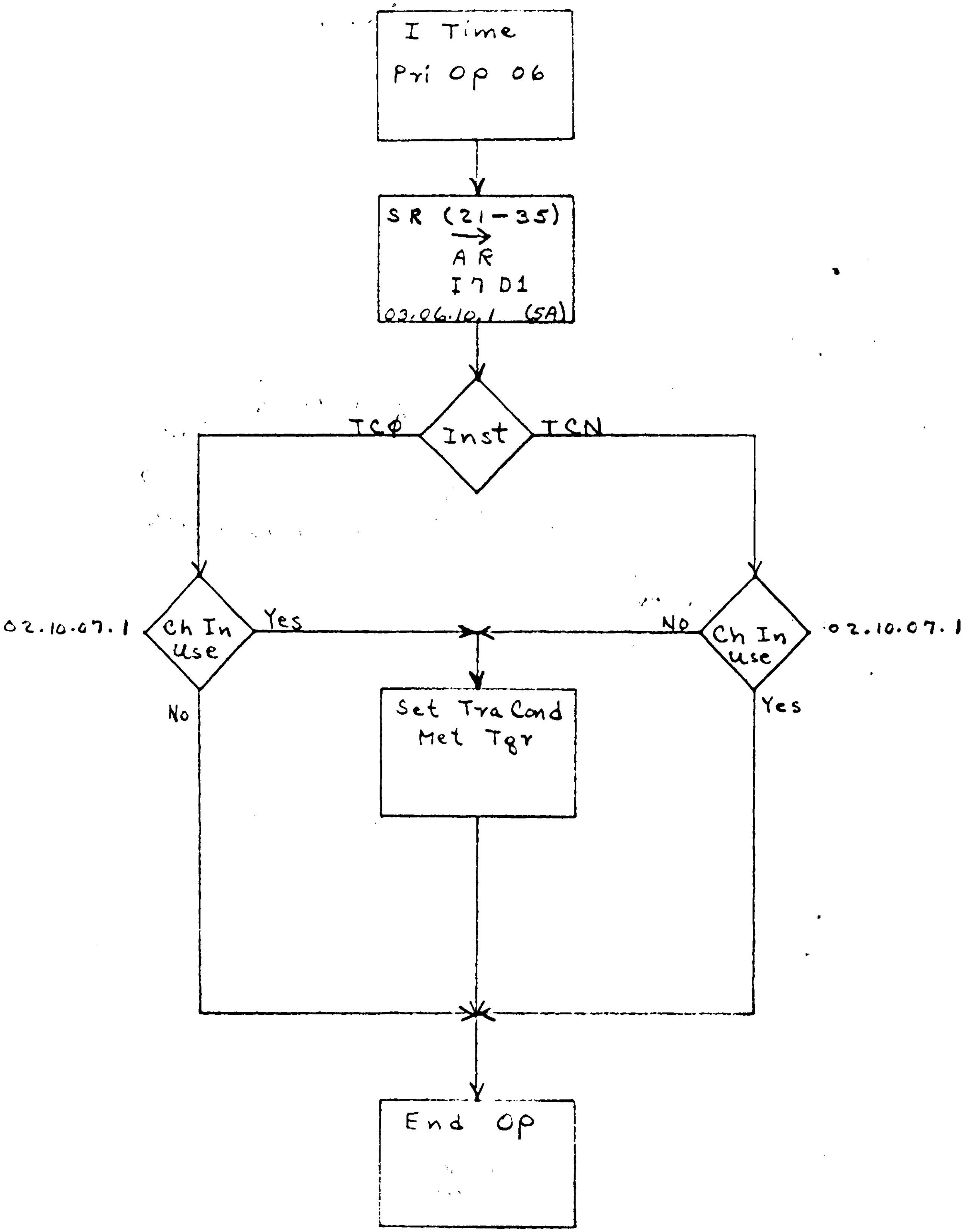


Fig 502-5

P. 500.1401

Revised 11-1-61 WEK

new

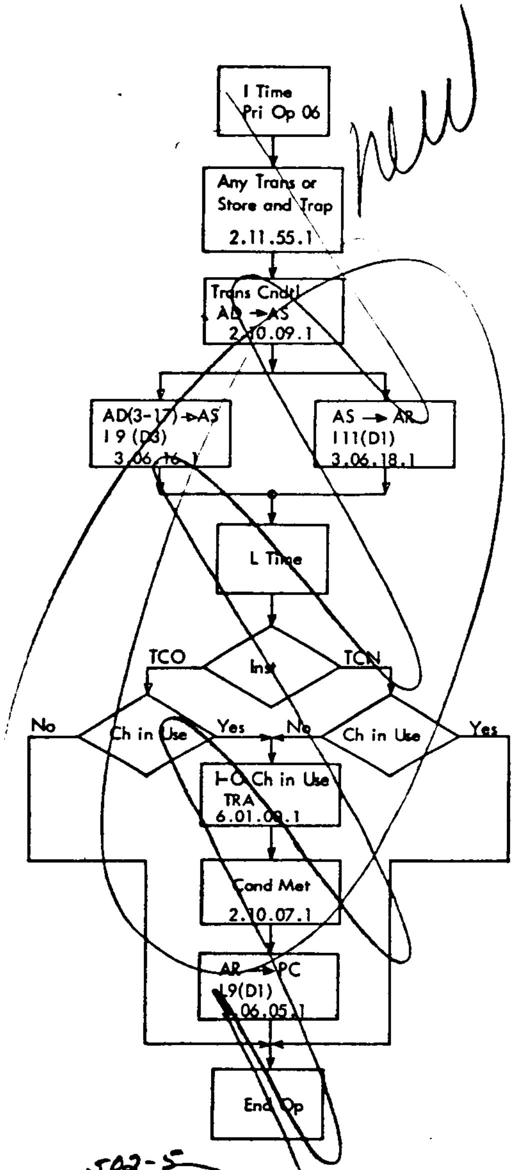


FIGURE ~~5.3-21~~ 502-5. TCO + 0060; + 0061; + 0062; Etc.
TCN - 0060; - 0061; - 0062; Etc.

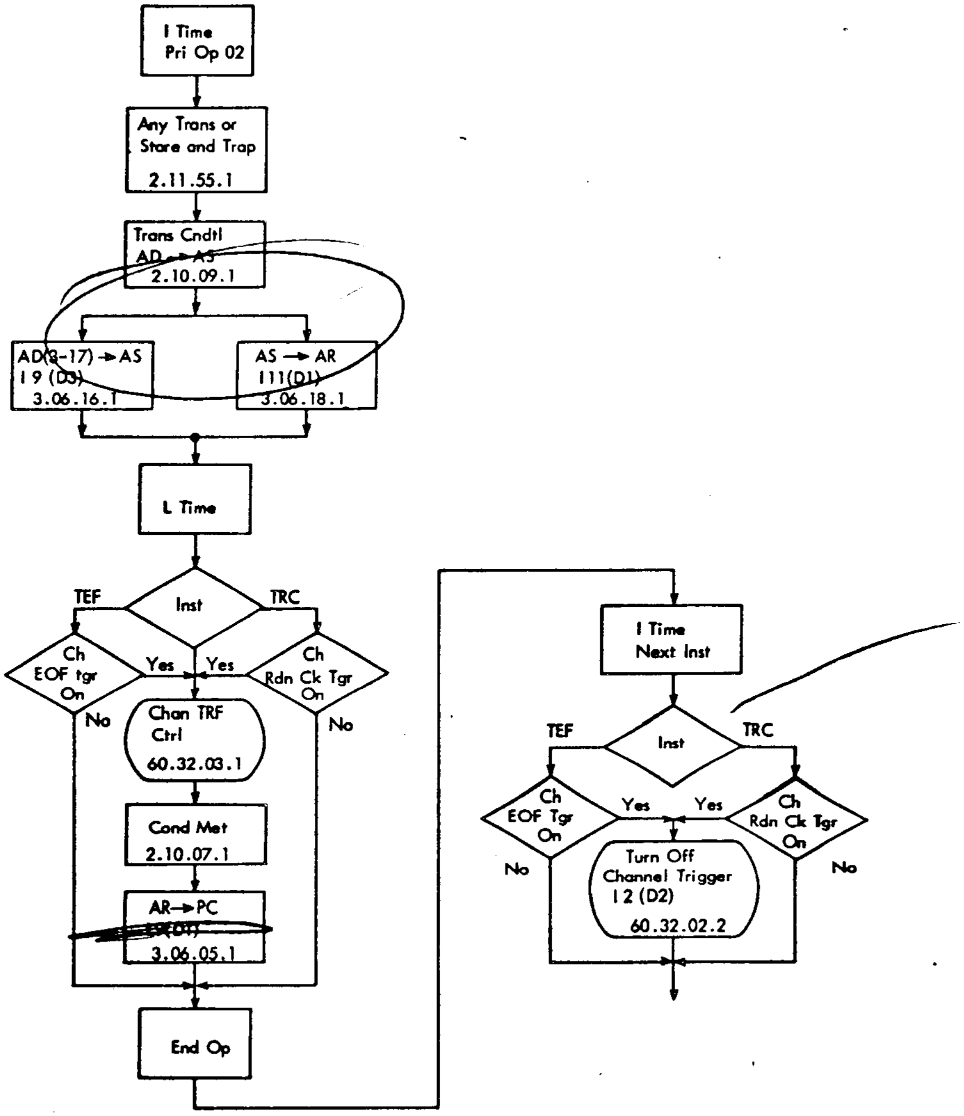


FIGURE ~~5.3-02~~ 502-6. TRC + 0022; -0022; + 0024; Etc.
TEF + 0030; -0030; + 0031; Etc.

~~50~~ 500.15

Transfer on Data Channel Redundancy Check TRC ±XXXX(I, L) Figure ~~5.3-32~~ 502-6

If the data channel redundancy check trigger is on, a transfer is taken to location X and the trigger is turned off. If the trigger is off, the computer takes the next sequential instruction. Operation codes +0022, -0022, +0024, -0024, +0026, -0026, +0027, and -0027 are used to select data channels A through H, respectively.

Transfer on Data Channel End of File TEF ±XXXX(I, L) Figure ~~5.9-32~~ 502-6

If the data channel end-of-file trigger is on, a transfer is taken to storage location X and the trigger is turned off. If the trigger is off, the computer takes the next instruction in sequence. Operation codes +0030, -0030, +0031, -0031, +0032, -0032, +0033, and -0033 are used to select data channels A through H, respectively.

5041

~~5.3.07~~ Skip Instructions

The skip instructions allow the programmer to alter the program to meet special conditions without stopping the computer. These instructions are similar to the conditional transfer instructions but, instead of transferring, they cause one or two instructions to be skipped. Skipping is accomplished by supplying an extra advance pulse to the instruction counter.

Most skip instructions cause the computer to skip when the condition being tested is met. The exceptions are the error testing and I-O testing instructions, which cause skipping when the condition being tested is not met; e.g., when there are no errors. This exception (skip on no error) allows a straight-line program until an error is detected. To process an error, an instruction transferring to an error subroutine usually follows the'test instruction. Another exception, the CAS instruction, has three possible results: it can fail to skip for AC greater, skip once for equal, or skip twice for AC less.

P Bit Test PBT -0760...0001(I, L) DLA Figure 504-1 ~~5-3-35~~

A bit in accumulator (P) position causes the computer to skip one instruction. If there is no bit in (P), the computer takes the next instruction in sequence.

Low-Order Bit Test LBT +0760...0001(I, L) DLA Figure 504-1 ~~5-3-35~~

A bit in accumulator (35) causes the computer to skip one instruction. If there is no bit in (35), the computer takes the next instruction in sequence.

Storage Zero Test ZET +0520(I, E) TLA DLA Figure 504-2 ~~5-3-36~~

If the contents of storage location X, except the sign, are zero, the computer will skip one instruction. If storage is not zero, the computer proceeds to the next instruction in sequence. Storage is unchanged. The information from storage on the SB is tested for zero as shown on Systems 2.12.52.1.

Storage Non-Zero Test NZT -0520(I, E) TLA DLA Figure 504-2 ~~5-3-36~~

If positions (1-35) of storage location X are not zero, the computer skips the next instruction. If the contents of storage location X are zero, no skip is taken. Storage is unchanged.

Compare Accumulator with Storage CAS +0340(I, E, L) TLA DLA Figure 504-3 ~~5-3-37~~

The accumulator is compared with the word at storage location X. Comparison is accomplished by taking an algebraic difference. If the accumulator is greater than the word in storage, no skip is taken. If the accumulator is equal to the word in storage, one instruction is skipped. If the accumulator is less than the word in storage, the next two instructions are skipped. Neither the accumulator nor the word in storage is changed.

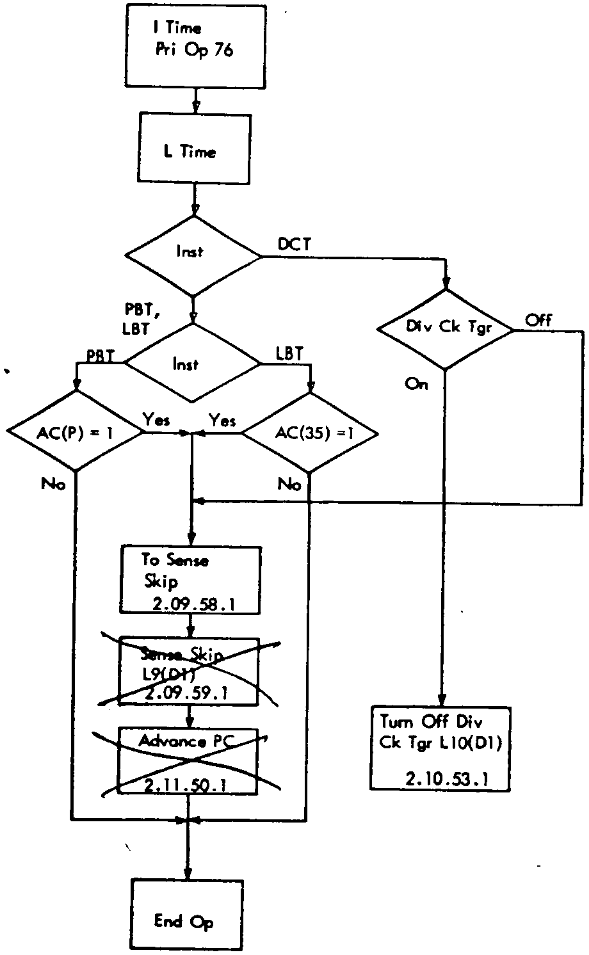


FIGURE ~~5-3-3~~5. 504-1 PBT -0760...0001; LBT +0760...0001; DCT +0760...0012

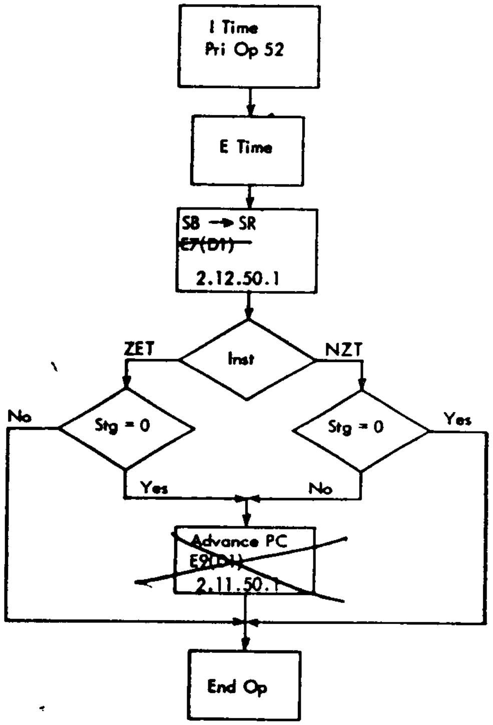
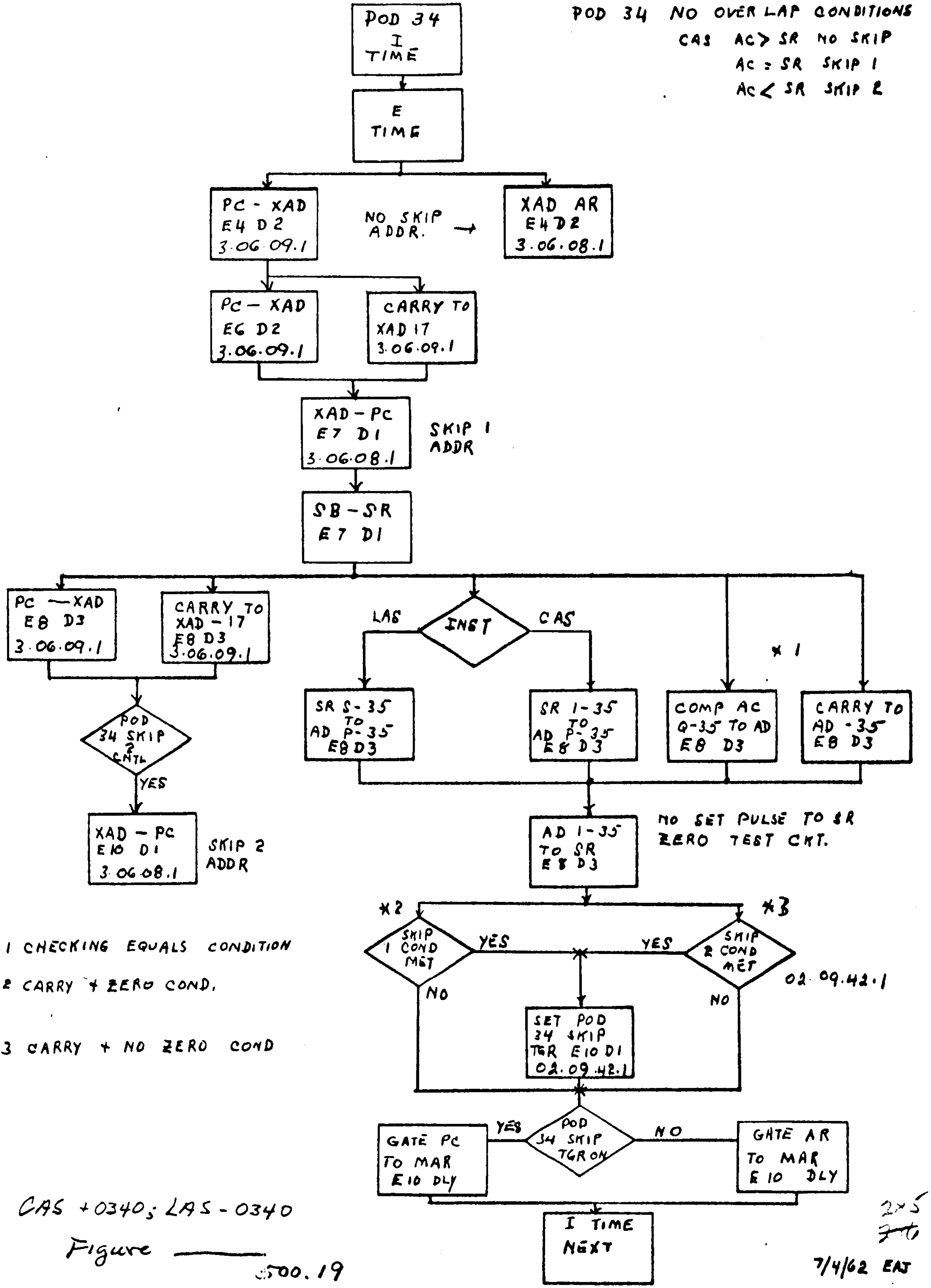


FIGURE ~~5-3-36~~. 504-2 ZET +0520; NZT -0520



POD 34 NO OVERLAP CONDITIONS

CAS AC > SR NO SKIP

AC = SR SKIP 1

AC < SR SKIP 2

*1 CHECKING EQUALS CONDITION

*2 CARRY + ZERO COND.

*3 CARRY + NO ZERO COND

CAS +0340; LAS - 0340

Figure ———— 500.19

2-5
~~2-6~~

7/4/62 EAJ

CAS and LAS Skip Tgr Requirements

Skip One Tgr

INST	SIGNS	SR AC SIGNS	P CARRY	ADP - 35 = 0 or $\overline{0}$
CAS	ALIKE	———	YES	0
LAS	———	———	YES	0

Skip Two Tgr

CAS	ALIKE	+		YES	$\overline{0}$
CAS	ALIKE		−	NO	
CAS	UNLIKE	+			
LAS				YES	$\overline{0}$

AC > STORAGE No Skip

AC = STORAGE SKIP ONE

AC < STORAGE SKIP TWO

500.19A

~~500.19C~~

mrs 5/18/62

To execute this instruction, an E cycle is required to bring the word in storage to the storage register; then an L cycle is used for two comparisons in the adders. For the first comparison, the SR and the complement of the AC are fed to the adders; the Q carry and sign conditions are matched to condition the first possible skip. The second comparison is made after a one has been added to the difference, to differentiate words of equal magnitude.

The following illustrate some of the possible combinations:

Number in Accumulator	Number in Storage	Q Carry Tgr 1st Comp	Q Carry Tgr 2nd Comp	Result
-6	-7	On	On	Next Instruction
-6	-6	Off	On	Skip 1 Instruction
-6	-4	Off	Off	Skip 2 Instructions
-6	+6	Off	On	Skip 2 Instructions
-0	+0	Off	On	Skip 2 Instructions
+0	-0	Off	On	Next Instruction
+6	-6	Off	On	Next Instruction
+6	+4	Off	Off	Next Instruction
+6	+6	Off	On	Skip 1 Instruction
+6	+7	On	On	Skip 2 Instructions

Logical Compare Accumulator with Storage LAS -0340(I, E, L) Figure ~~5.3-37~~ 504-3
TLA - DLA

This instruction compares the contents of the AC(P, 1-35) with the logical word (S, 1-35) stored at location X. The sign of the AC is disregarded; the contents of the AC and storage are unchanged.

If the contents of the AC are greater than the contents of storage location X, the computer takes the next instruction in sequence. If the AC equals storage, the computer skips one instruction. If the contents of the AC are less than the contents of storage, the computer will skip the next two instructions.

This instruction is executed the same as CAS, except that the signs are not used for an algebraic comparison and AC(P) is compared against SR(S).

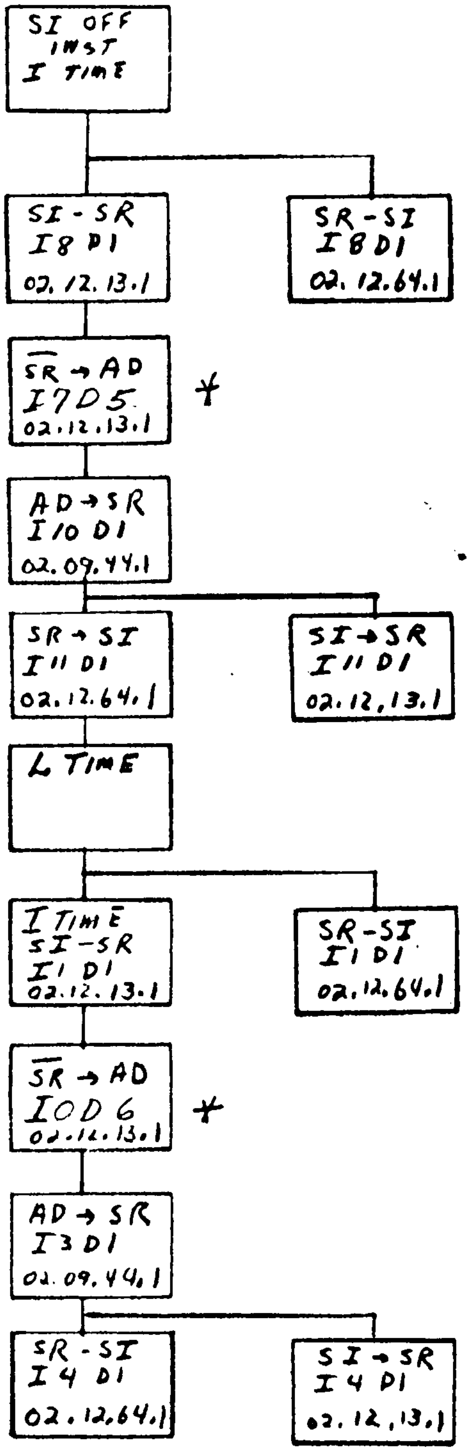
Plus Sense PSE +0760...XXXX(I, L) DLA

This instruction provides a means to test the status of any of the six sense switches, to turn on or off the four console sense lights, and to permit the transmission of an impulse to or from the exit or entry hubs of either the printer or punch.

The address portion of the instruction determines whether a light, switch, printer, or card punch is being sensed; further, it determines which light, switch, or hub is being sensed. The octal addresses for the different sense instructions are:

Address		Instruction
0140	SLF	Turn off all sense lights (Figure ~~5.3-38~~ 504-4)
0141-0144	SLN	Turn on sense light 1, 2, 3, or 4, respectively (Figure ~~5.3-39~~ 504-5)

I TIME SENSE OFF INDICATOR INST.

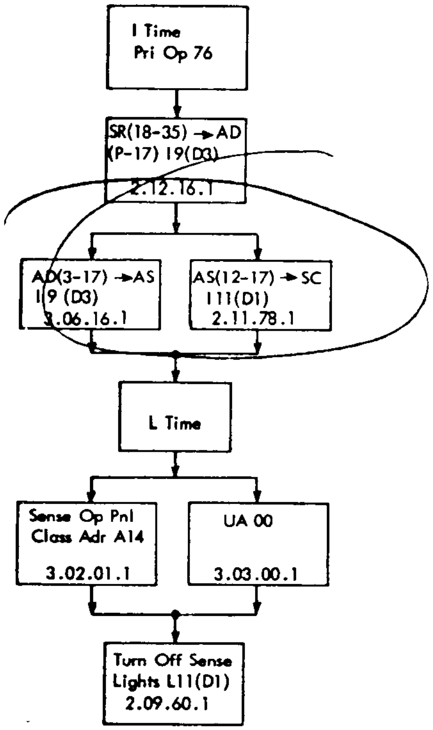


+ TIMING DETERMINED BY SENSE OFF DECODE LINE WHICH IS POD NOT "L" TIME

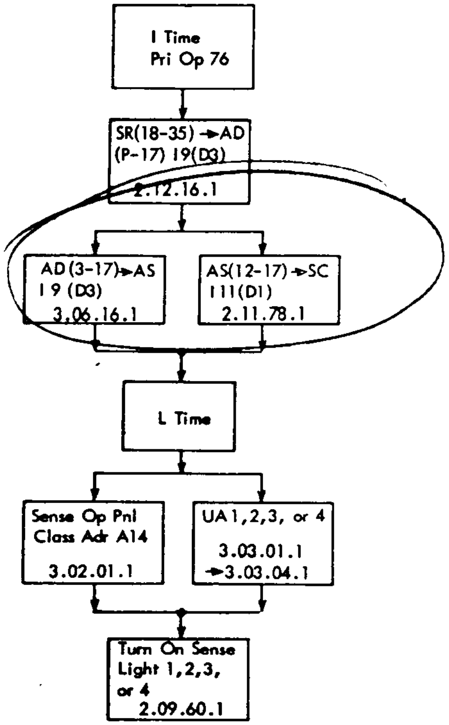
288 ~~[illegible]~~

500.20A

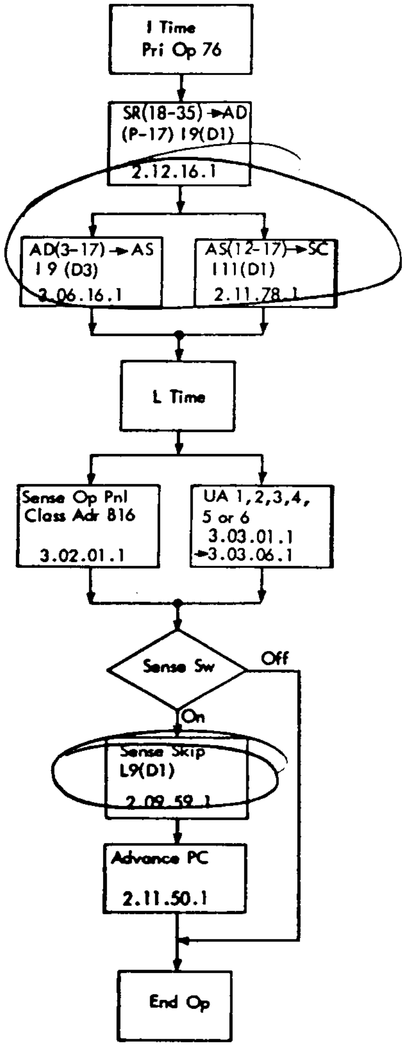
MFS 5/17/62 ✓



504-4
FIGURE ~~5.3-38~~. √PSE +0760...0140
SLF



504-5
FIGURE ~~5.3-39~~. √PSE +0760...0141, 0142, 0143, 0144
SLN



504-6
FIGURE ~~5.3-40~~. √PSE +0760...0161, 0162, ETC.
SWT

Address		Instruction
0161- 0166	SWT	If the corresponding sense switch is down (on), the computer skips the next instruction. If the sense switch is up (off), the computer takes the next instruction in sequence (Figure ~~5.3-40~~ 504-6).
1341- 1342 2341- 2342 3341- 3342 4341- 4342 5341- 5342 6341- 6342 7341- 7342 10341-10342	SPU	The computer causes an impulse to appear at the specified exit hub of the control panel of the card punch attached to Data Channel A, B, C, D, E, F, G, or H respectively (Figure ~~5.3-41~~). 504-7
1360, 2360 3360 4360 5360 6360 7360 10360	SPT	If an impulse is present at the entry hub of the control panel of the printer attached to Data Channel A, B, C, D, E, F, G, or H respectively, the computer skips the next instruction. If there is no impulse, the computer takes the next instruction in sequence (Figure ~~5.3-42~~). 504-8
1361- 1372 2361- 2372 3361- 3372 4361- 4372 5361- 5372 6361- 6372 7361- 7372 10361-10372	SPR	The computer causes an impulse to appear at the specified exit hub of the control panel of the printer attached to Data Channel, A, B, C, D, E, F, G, or H, respectively (Figure ~~5.3-43~~). 504-9

Minus Sense SLT MSE -0760...XXXX(I, L) DLA Figure ~~5.3-44~~ 504-10

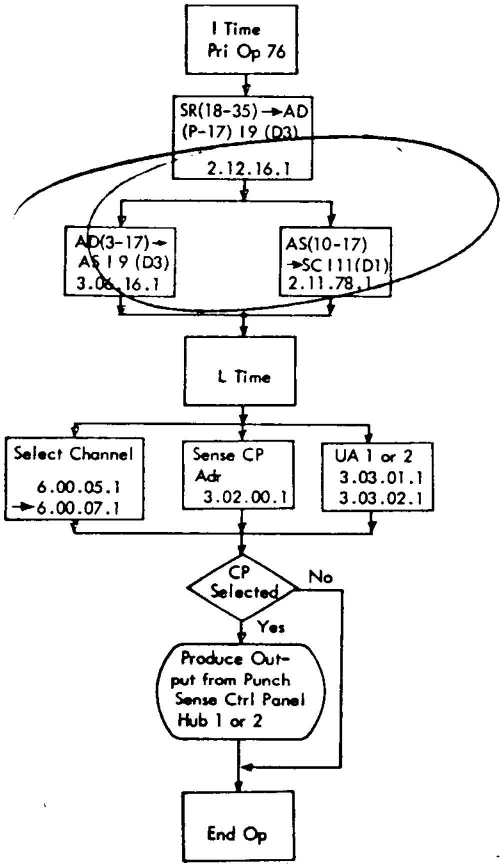
If the sense light, on the operator's console, corresponding to the address portion of the instruction is on, this light is turned off and the computer skips the next instruction. If the sense light is off the computer takes the next instruction in sequence. Addresses 0141-0144 correspond to sense lights 1-4, respectively.

Input-Output Check Test IOT +0760...0005(I, L) DLA Figure 5~~.3-45~~ 504-11

If the I-O check trigger is on, the indicator is turned off and the computer takes the next instruction in sequence. If the I-O check indicator is off, the computer skips the next instruction.

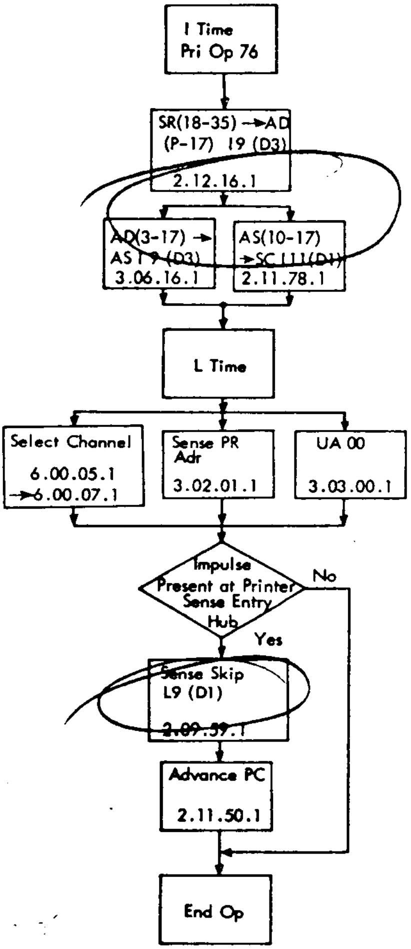
Divide Check Test DCT +0760...0012 (I, L) DLA Figure ~~5.3-35~~ 504-1

This instruction examines the status of the divide check trigger. If the trigger is off, the next instruction is skipped. If the trigger is on, it is turned off and the computer takes the next instruction in sequence.



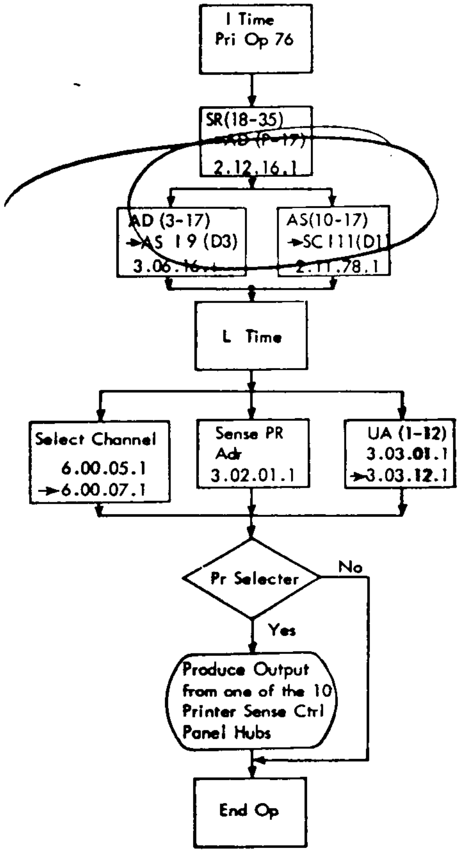
SPU

FIGURE ~~5-341~~ 504-7. PSE +0760...1341, 2342, 3341, ETC.



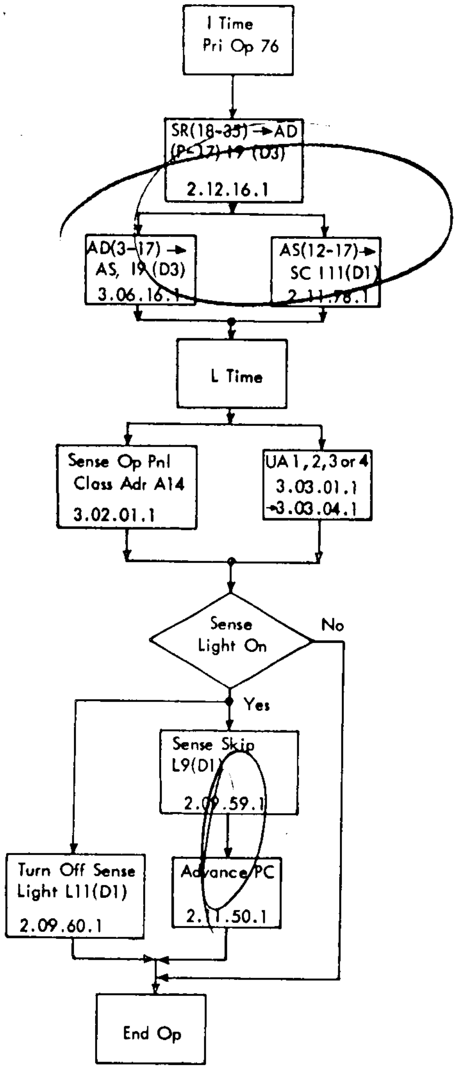
SPT

FIGURE ~~5-342~~ 504-8. PSE +0760...1360, 2360, ETC.



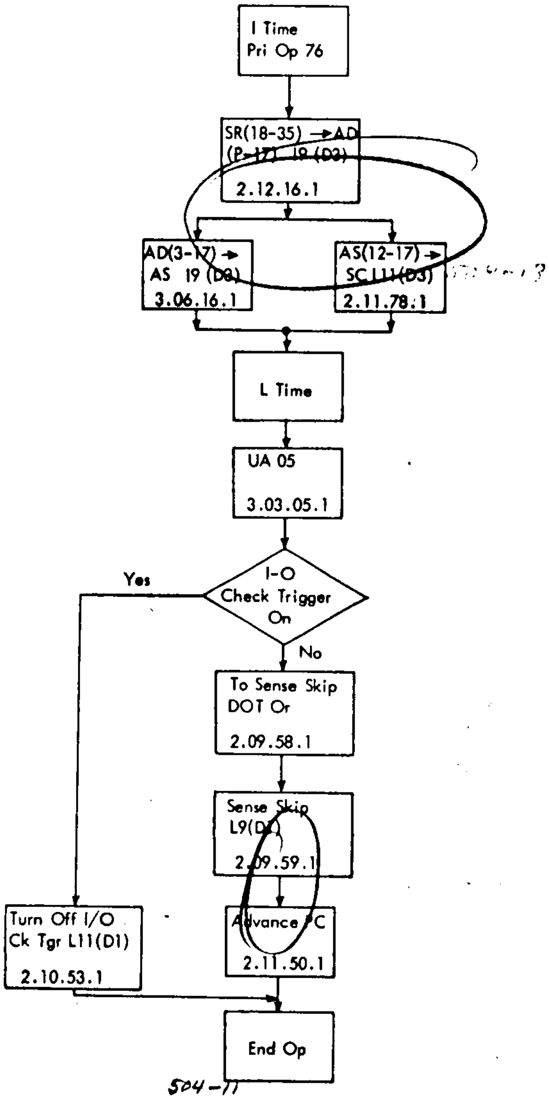
SPR

FIGURE ~~5-343~~ 504-9. PSE +0760...1361, 1372, 2361, 2362, ETC.



504-10
FIGURE ~~5-8-44~~. MSE -0760...0141, 0142, 0143, 0144

SLT



504-11
FIGURE ~~5-8-45~~. IOT +0760...0005

~~116~~
500.24

312

Beginning-of-Tape Test BTT +0760...XXXX (I, L) DLA Figure ~~5.3.46~~ 504-12

This instruction tests the status of the beginning-of-tape indicator in a particular data channel. Address 1000-10000 selects data channels A-H respectively. If the beginning-of-tape indicator for the selected channel has been turned on by a previous instruction, the indicator is turned off and the computer takes the next instruction in sequence. If the indicator is already off, the computer skips the next instruction.

End-of-Tape Test ETT -0760...XXXX (I, L) DLA Figure ~~5.3.46~~ 504-12

This instruction uses address 1000-10000 to select the data channel in which the end-of-tape indicator is to be tested. If the indicator is on, it is turned off and the computer takes the next instruction in sequence. If the indicator is off, the computer skips the next instruction.

~~5.3.08~~ 505 Control Instructions

Control instructions are provided so the programmer may change the problem conditions or service the computer.

Halt and Proceed HPR +0420 (I, L) Figure ~~5.3.47~~ 505-1

This instruction causes the computer to stop at the end of I time. When the start button is depressed, the computer proceeds to the next instruction in sequence. When halted, the PC contains the address of the next sequential instruction.

Halt and Transfer HTR +0000 (I, L) DLA Figure ~~5.3.48~~ 505-2

This instruction causes the computer to stop at the end of I time by turning on the master stop trigger at I_{10}(D1). Depressing the start button causes the computer to proceed in L time of a transfer operation and the computer takes an instruction transfer to location X. When halted, the PC contains the address of the HTR instruction.

No Operation NOP +0761 (I, L)

The NOP instruction performs no active function, but is used to reserve space for other instructions. Since this instruction has a primary operation 76, an I and an L cycle are required. The only function of this instruction is to turn on the end operation trigger to allow the computer to proceed to I time of the next sequential instruction. SOD 01 on Systems 3.07.01.1 causes L END OP on Systems 8.00.09.1.

Execute XEC +0522 (I) DLA or TLA Figure ~~5.3.49~~ 505-3

This instruction causes the computer to perform the instruction at location X. The program counter is not altered; therefore, after the instruction at location X has been executed, the computer proceeds to the next sequential instruction (instruction in next position beyond the XEC instruction). ~~XEC prevents AR to PC on Systems 3.06.05.1.~~

If the instruction at location X is an unconditional transfer or a successful conditional transfer, the computer does not proceed to the next instruction following the XEC, but proceeds to the address specified by the transfer. Another exception occurs when the instruction at location X is a skip type, and the conditions are met for this skip. The PC is stepped and the skip is in relation to the location of the XEC instruction.

Set Sign Plus SSP +0760...0003 (I, L) DLA Figure ~~5.3.50~~ 505-4

This instruction places a zero (a plus) in the accumulator sign position. Positions (Q-35) of the accumulator are unchanged.

Set SC
Sec Op 00
UA 00

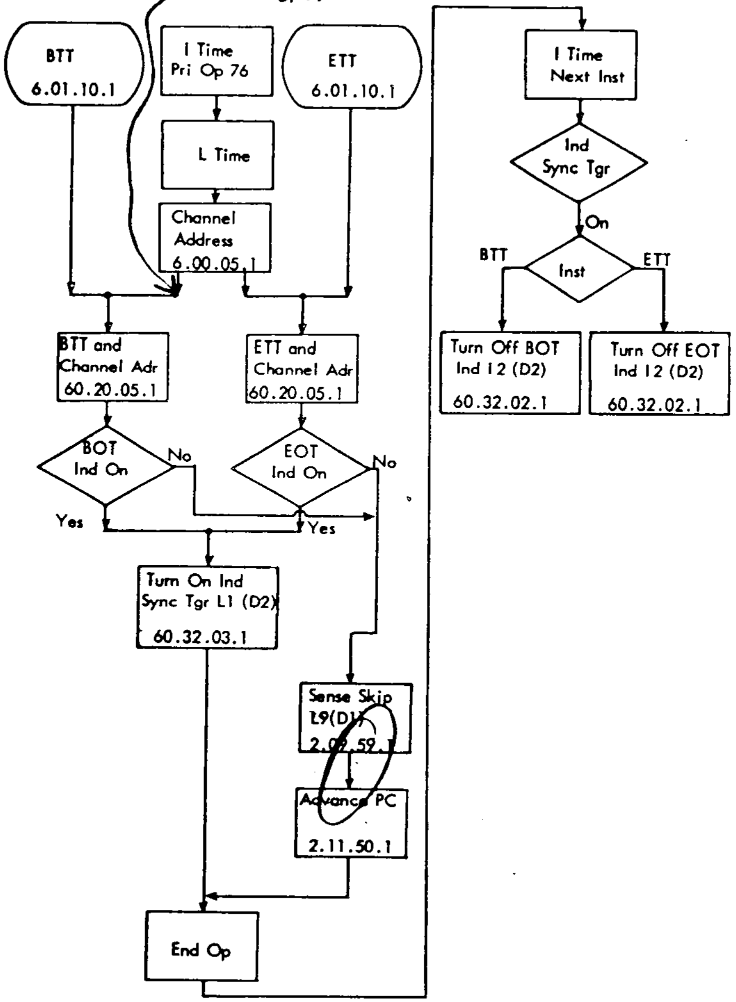


FIGURE ~~5-2-46~~ 504-12. BTT +0760, ETT -0760

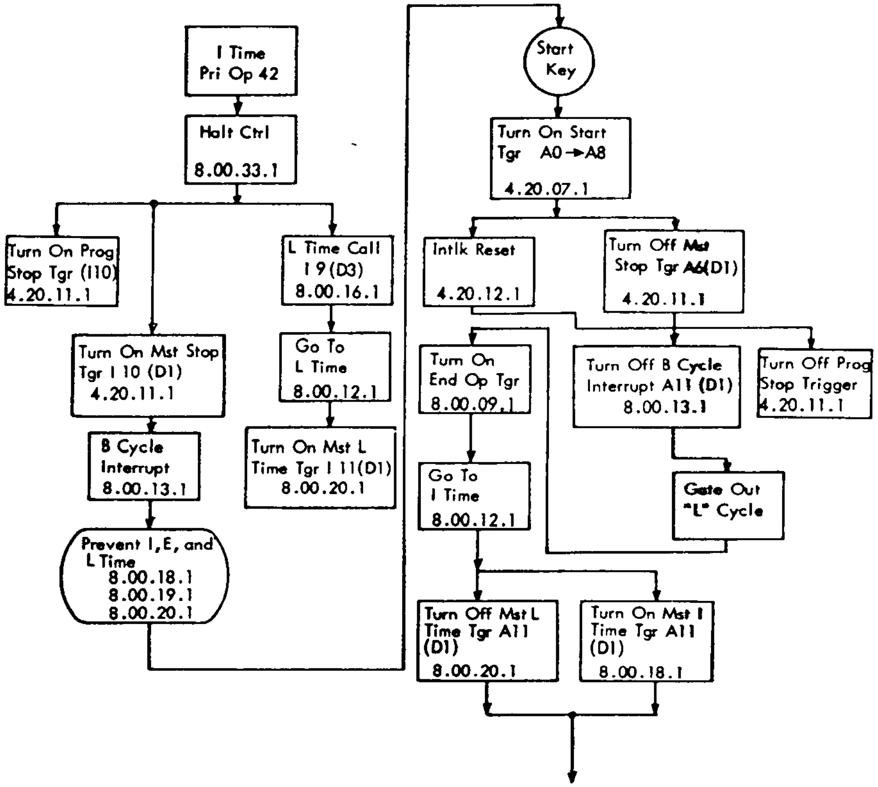


FIGURE ~~5-2-47~~ 505-1. HPR +0420

~~108~~
500.26

~~315~~
311

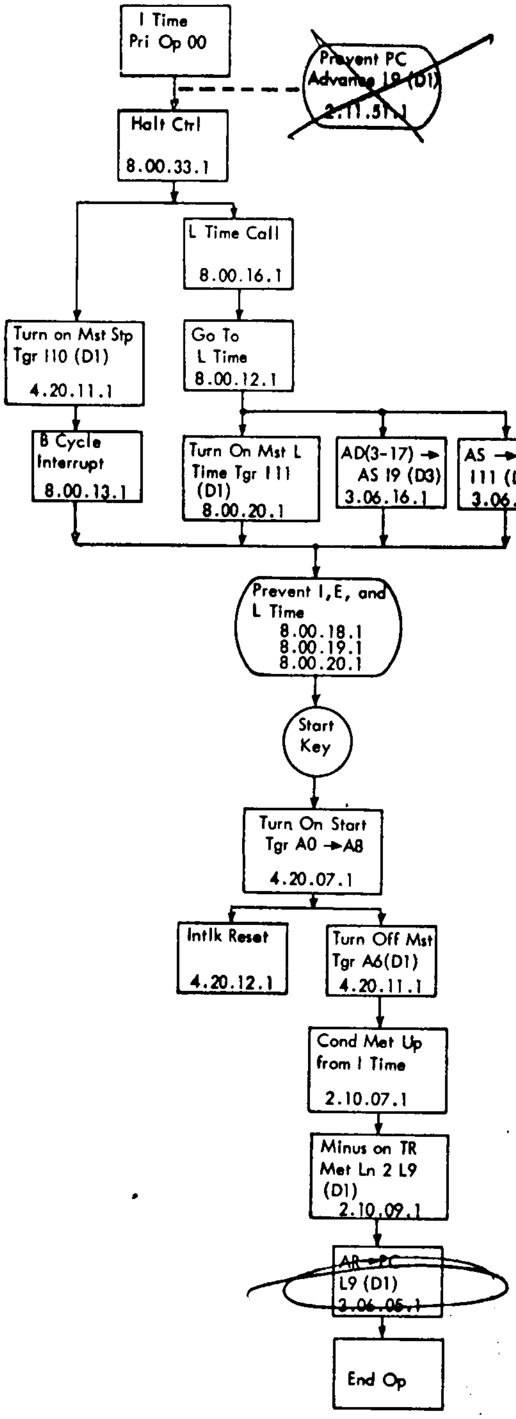
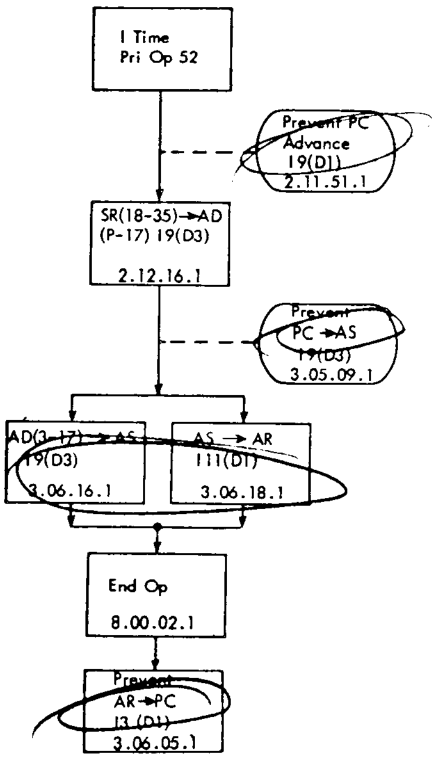
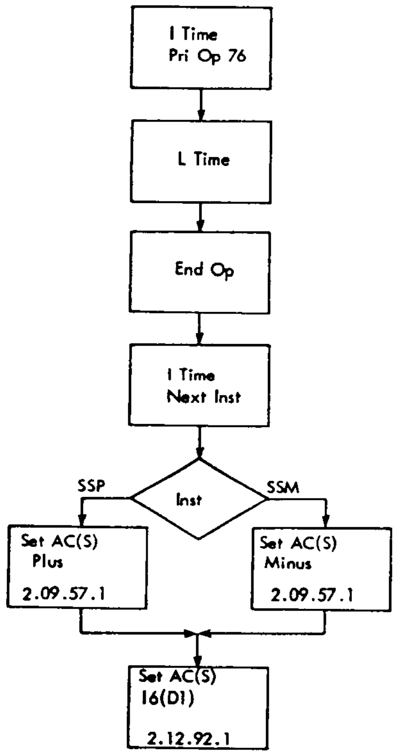


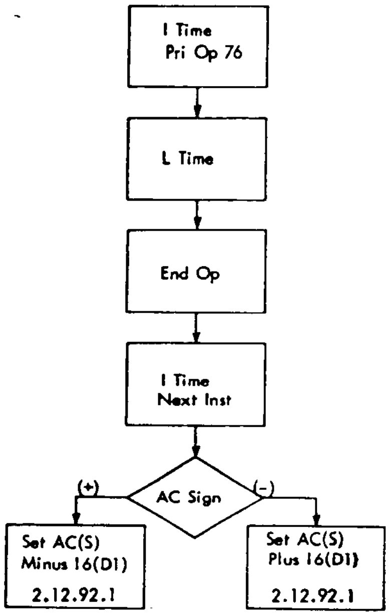
FIGURE ~~5.8-48.~~ 505-2 HTR + 0000



505-3
FIGURE ~~5.3.49~~. XEC + 0522



505-4
FIGURE ~~5.3.50~~. SSM - 0760...0003; SSP + 0760...0003



505-5
FIGURE ~~5.3.51~~. CHS +0760...0002

Set Sign Minus SSM -0760...0003(I, L) Figure ~~5.3-50~~ 505-4 DLA

This instruction places a one (a minus) in the accumulator sign position. Positions (Q-35) of the accumulator are unchanged.

Change Sign CHS +0760...0002(I, L) Figure ~~5-3-51~~ 505-6 DLA

This instruction complements the sign position of the accumulator. A one is replaced by a zero and a zero is changed to a one. Positions (Q-35) of the accumulator are unchanged.

506 ~~5.3.09~~ Sense Indicator Instructions

The sense indicator instructions are a group of instructions which operate on the sense indicator register. These instructions enable the computer to set and test the indicators under program control.

Load Indicators LDI +0441(I,E) DLA or TLA Figure 506-1 ~~5.3.52~~

The contents of storage location X (S,1-35) are placed in indicator positons (0-35). The contents of storage are unchanged.

Store Indicators STI +0604(I,E) SLA

The contents of indicator positions (0-35) replace the contents of storage location X. The indicators are unchanged. Execution of this instruction is the same as STO (Figure 3.5-1, Section 3.5.01) except that SI(0-35) is taken to the SR rather than AC (S,1-35). SI(0-35) to the SR(S,1-35) is shown on Systems 2.12.13.1.

OR Storage to Indicators OSI +0442(I,E) DLA or TLA Figure 506-1 ~~5.3.52~~

This instruction places the logical OR of the word at storage location X and the contents of the indicators in the sense indicator register. Storage is unchanged.

Invert Indicators from Storage IIS +0440(I,E) DLA or TLA Figure 506-1 ~~5.3.52~~

This instruction inverts the positions of the SI register which have corresponding "1" bits in the word at storage location X.

Reset Indicators from Storage RIS +0445(I,E) DLA or TLA Figure 506-2 ~~5.3.53~~

This instruction utilizes the word at storage location X to reset the sense indicator register position. A "1" bit in any position of the word causes the corresponding sense indicator trigger to be turned off. Indicator positions which correspond to "0" bits are unchanged. Storage is unchanged.

Set Indicators of Right Half SIR +0055(I) Figure 506-3 ~~5.3.54~~

For this instruction, the control field (18-35) of the instruction is OR'ed with the right half of the sense indicator register. A "1" bit in either the control field or the indicator places a "1" bit in the corresponding sense indicator. Because the only

IIS - Invert Indicators From Storage - +0440

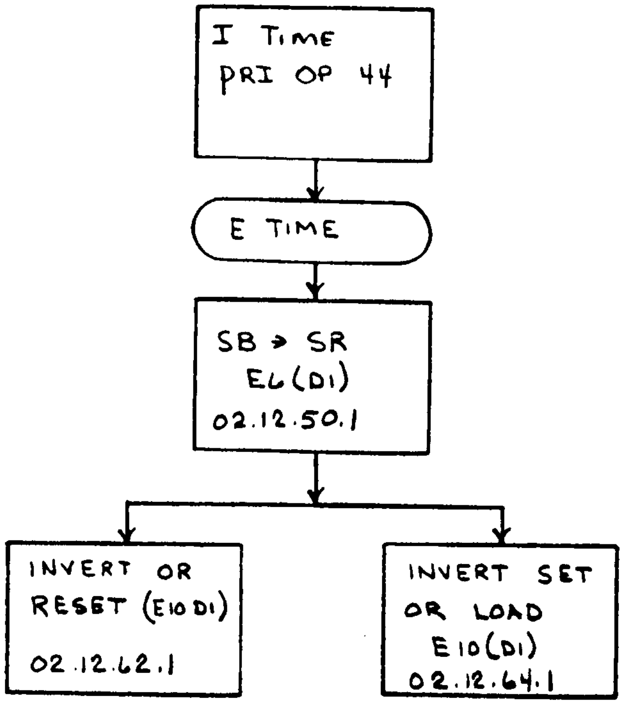


Figure 506-1

LDI- LOAD INDICATORS + 0441

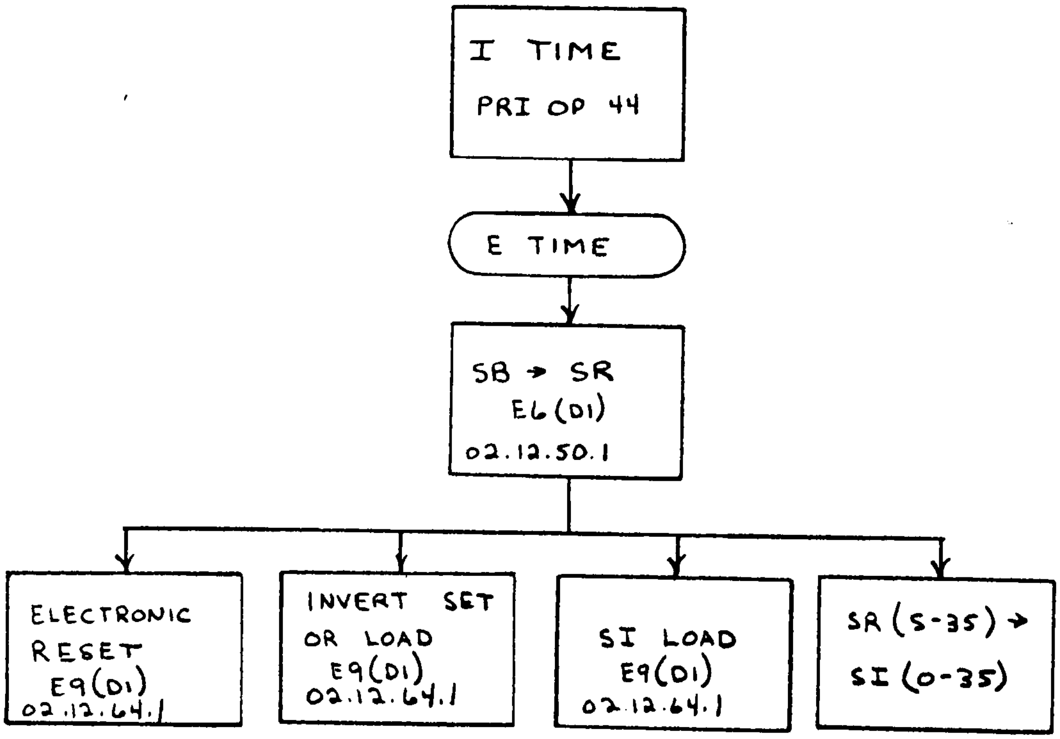


Figure 506-1A
page 500.31A

OSI- Or Storage To Indicators 0442
RIS - Reset Indicators From Storage 0445

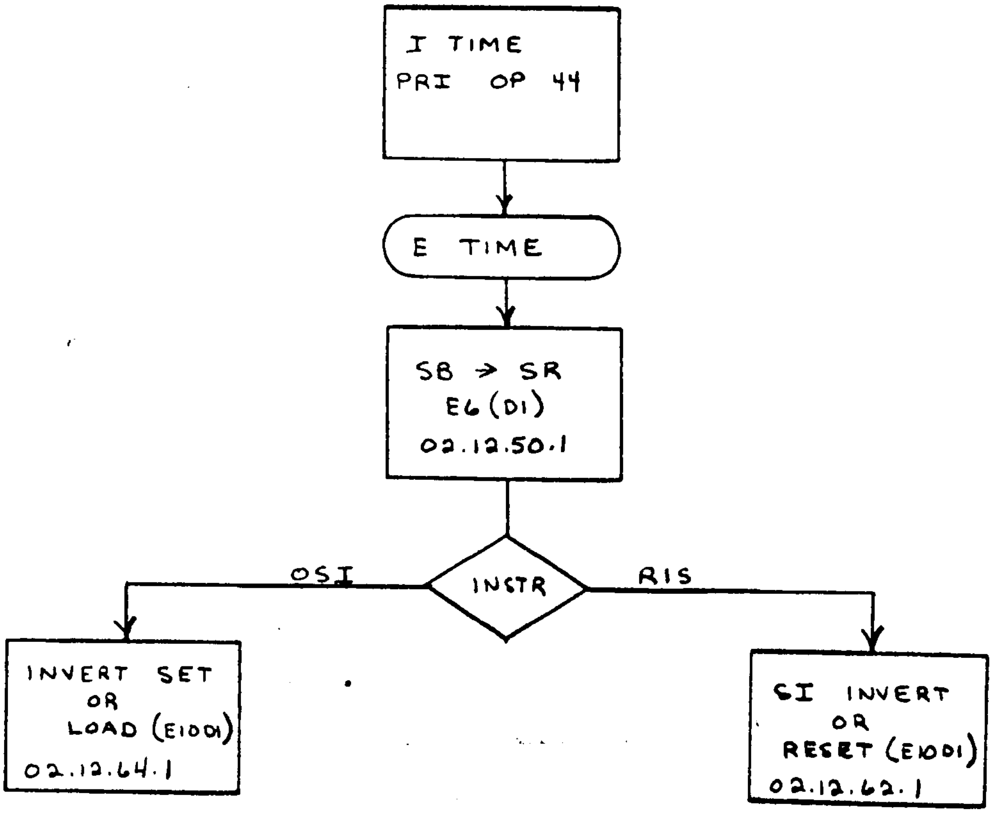
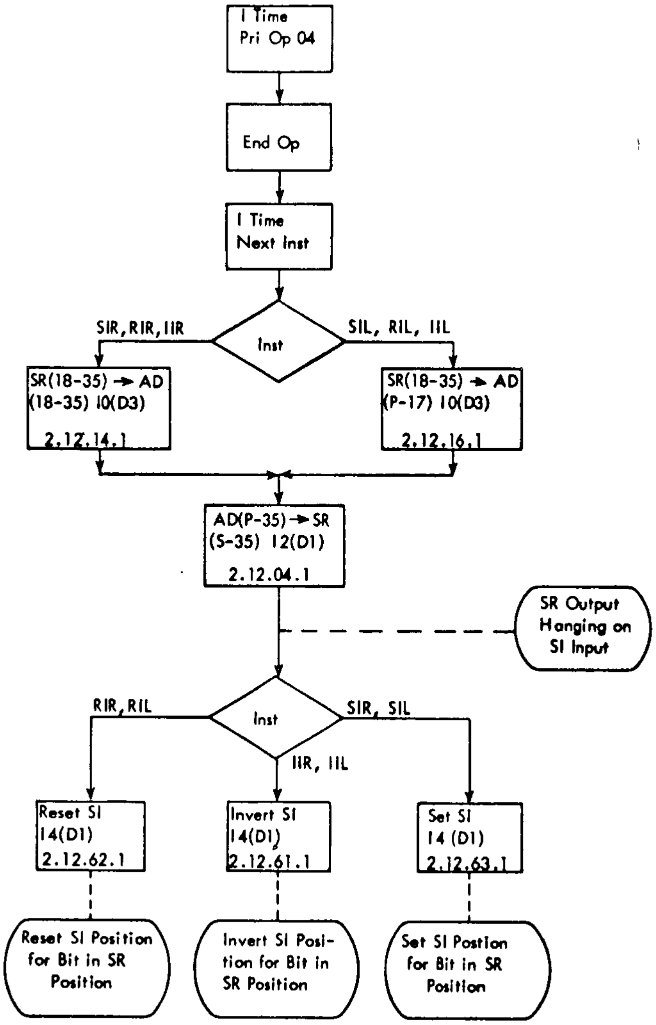


Figure 506-1B


I Time
Pri Op 04
End Op
I Time
Next Inst
Inst
SIR, RIR, IIR
SIL, RIL, IIL
SR(18-35) → AD (18-35) 10(D3)
2.12.14.1
SR(18-35) → AD (P-17) 10(D3)
2.12.16.1
AD(P-35) → SR (S-35) 12(D1)
2.12.04.1
SR Output Hanging on SI Input
Inst
RIR, RIL
SIR, SIL
IIR, IIL
Reset SI
14(D1)
2.12.62.1
Invert SI
14(D1)
2.12.61.1
Set SI
14 (D1)
2.12.63.1
Reset SI Position for Bit in SR Position
Invert SI Position for Bit in SR Position
Set SI Postion for Bit in SR Position

506-3

FIGURE ~~5.3-54~~ SIR + 0055; SIL - 0055; RIR + 0057; RIL - 0057; IIR + 0051; IIL - 0051

IIL - Invert Indicators Of Left Half 0051

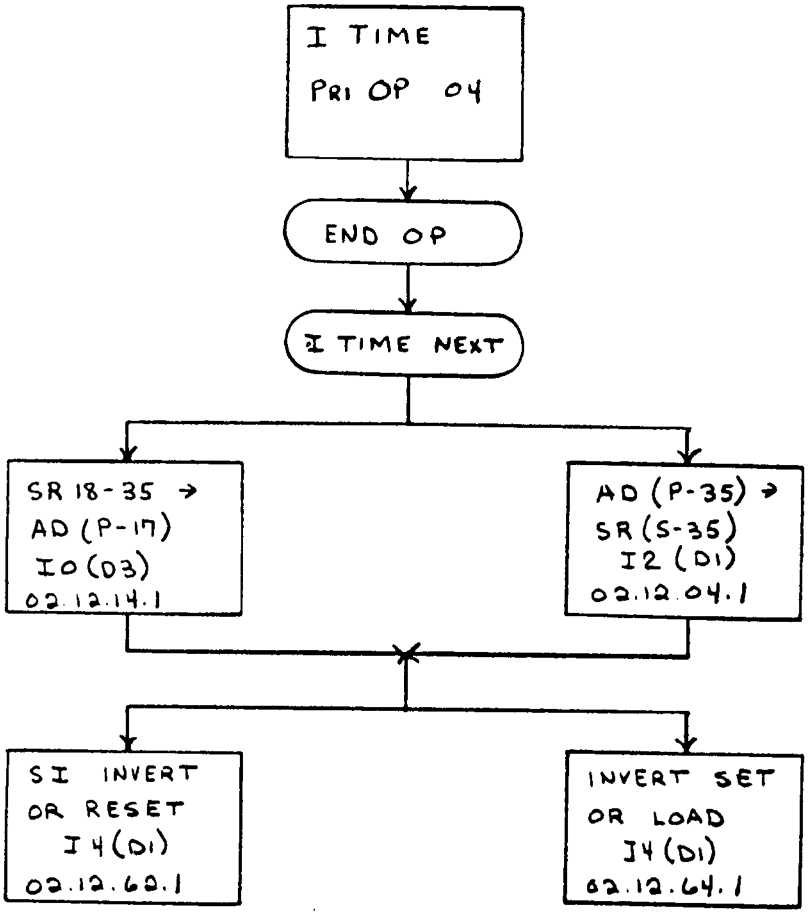


Figure 506-3A

RIR - Reset Indicators From Right Half - 0057

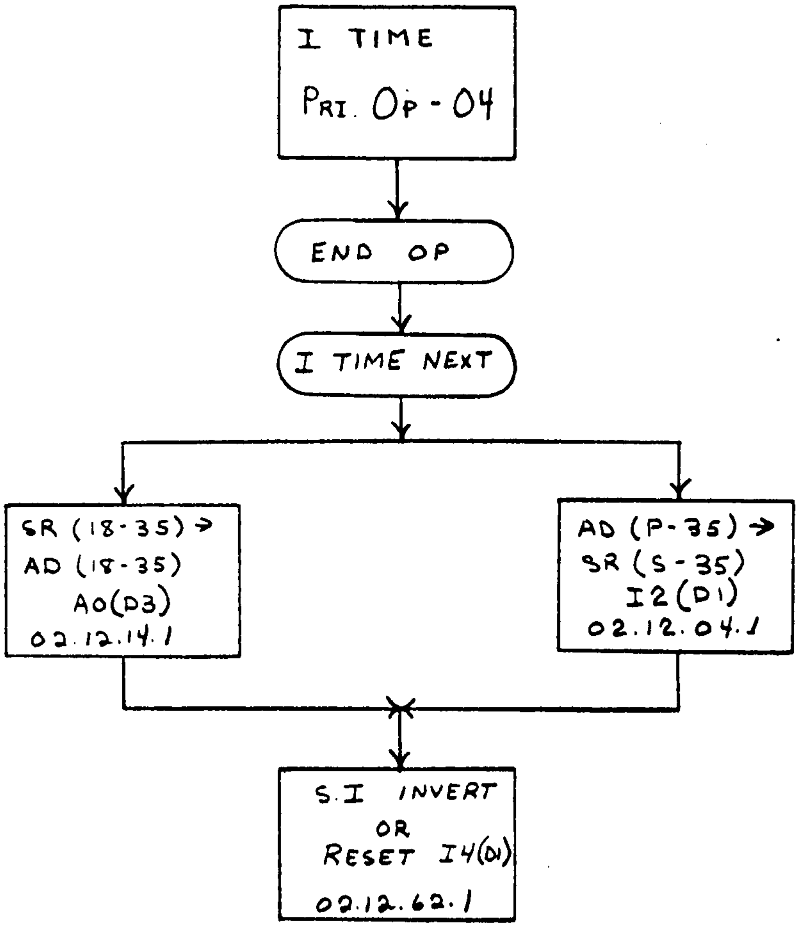


Figure 506-3B
page 500.32 B

input to the sense indicator register is from the SR, the control field must be placed in the right half of the SR, and the left half of the SR must be cleared before setting the indicators. This is accomplished by gating only the right half of the SR to the adders and then gating adders (P, 1-35) to SR(S, 1-35).

Set Indicators of Left Half SIL -0055 (I) Figure ~~5-2-54~~ 506-3

This instruction OR's the control field (18-35) of the instruction with the left half of the sense indicator register. Execution is the same as SIR except that the control field must be switched to the left half of the SR, and the right half of the SR must be cleared before the indicators can be set. This is done by routing SR(18-35) to AD(P-17).

Reset Indicators of Right Half RIR +0057 (I) Figure ~~5-3-54~~ 506-3

This instruction causes the reset of the positions of the right half of the sense indicator register which correspond to ones in the control field (18-35) of the instruction. Indicator positions corresponding to zeros are unchanged. Execution of this instruction is identical to SIR, except that the reset-indicators input is used instead of set indicators.

Reset Indicators of Left Half RIL -0057 (I) Figure ~~5-3-54~~ 506-3

For this instruction, the control field of the instruction is used as a reset mask for the left half of the indicator register. A one in the control field resets the corresponding indicator position. Execution of this instruction is similar to that of SIL except that the reset-indicators pulse is used.

Invert Indicators of Right Half IIR +0051 (I) Figure ~~5-3-54~~ 506-3

This instruction inverts positions of the right half of the indicator register which correspond to ones in the control field of the instruction. Indicator positions corresponding to zeros in the control field are unchanged. Execution is identical to that of SIR except that the invert-indicators pulse to the indicator input is used.

Invert Indicators of Left Half IIL -0051 (I) Figure ~~5-3-54~~ 506-3

This instruction inverts positions of the left half of the indicator register which correspond to ones in the control field (18-35) of the instruction. Indicator positions corresponding to zeros in the control field are unchanged. Execution of this instruction is identical to that of SIL, except that an invert-indicators pulse is used.

Place Indicator in Accumulator PIA -0046 (I) Figure ~~5-3-55~~ 506-4

The contents of sense indicators (0-35) are placed in positions (P, 1-35) of the accumulator. The sense indicators are unchanged. AC positions Q and S are cleared.

Place Accumulator in Indicators PAI +0044 (I) Figure ~~5-3-55~~ 506-4

The contents of the accumulator (P, 1-35) are placed in the sense indicator register. The accumulator is unchanged.

IIA - Invert Indicators From Accumulator + 0041

RIA - Reset Indicators From Accumulator - 0042

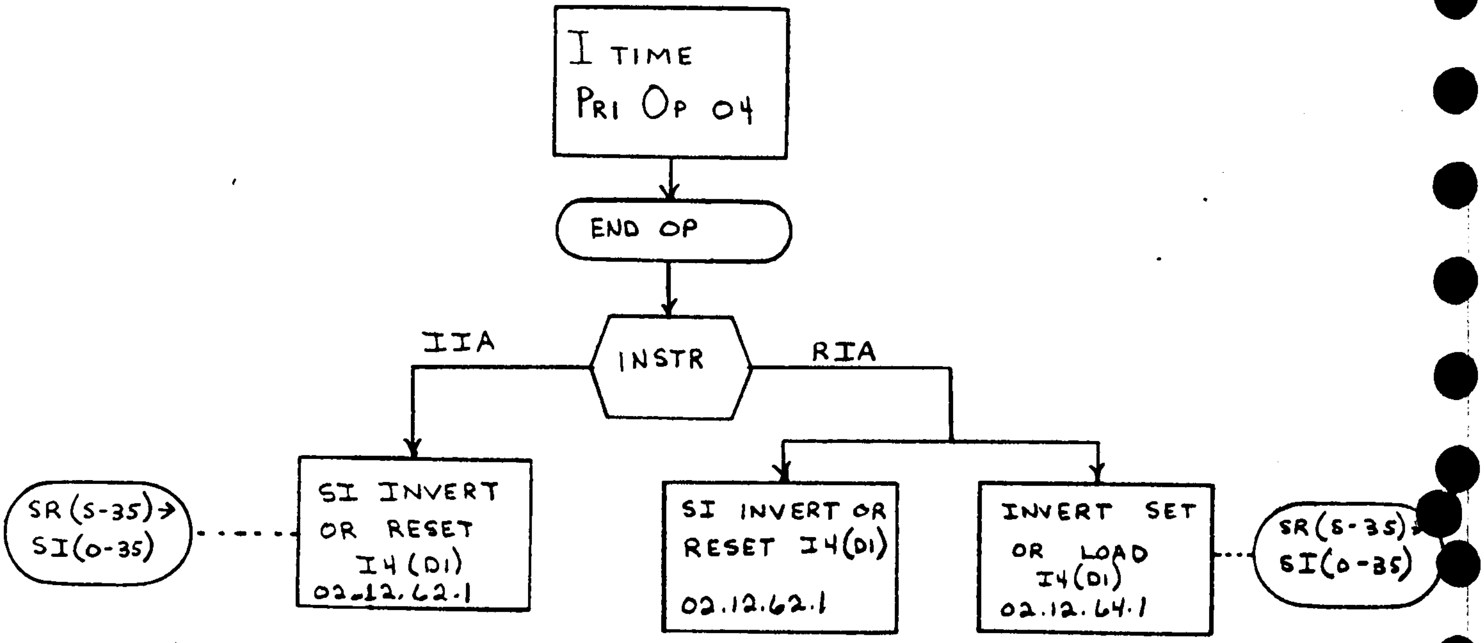


Figure 506 - 4
page 500.34

PAI- Place Accumulator In Indicators- +0044

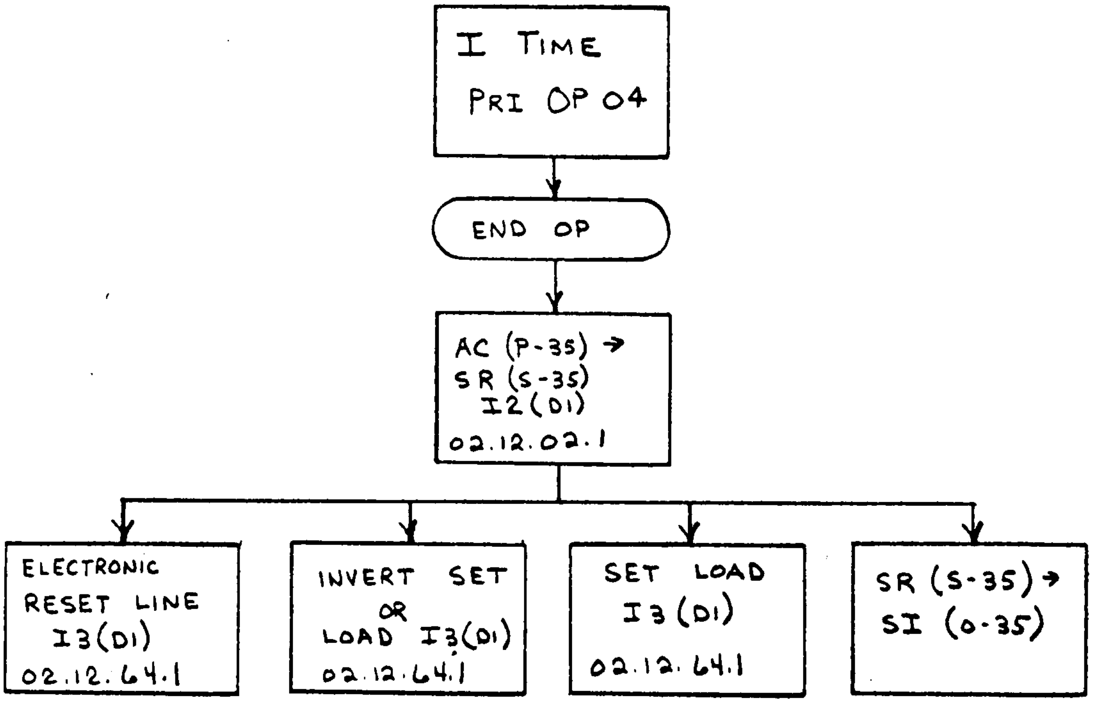


Figure 506-4A

PIA - Place Indicators In Accumulator — 0046
OAI - OR Accumulator To Indicators + 0043

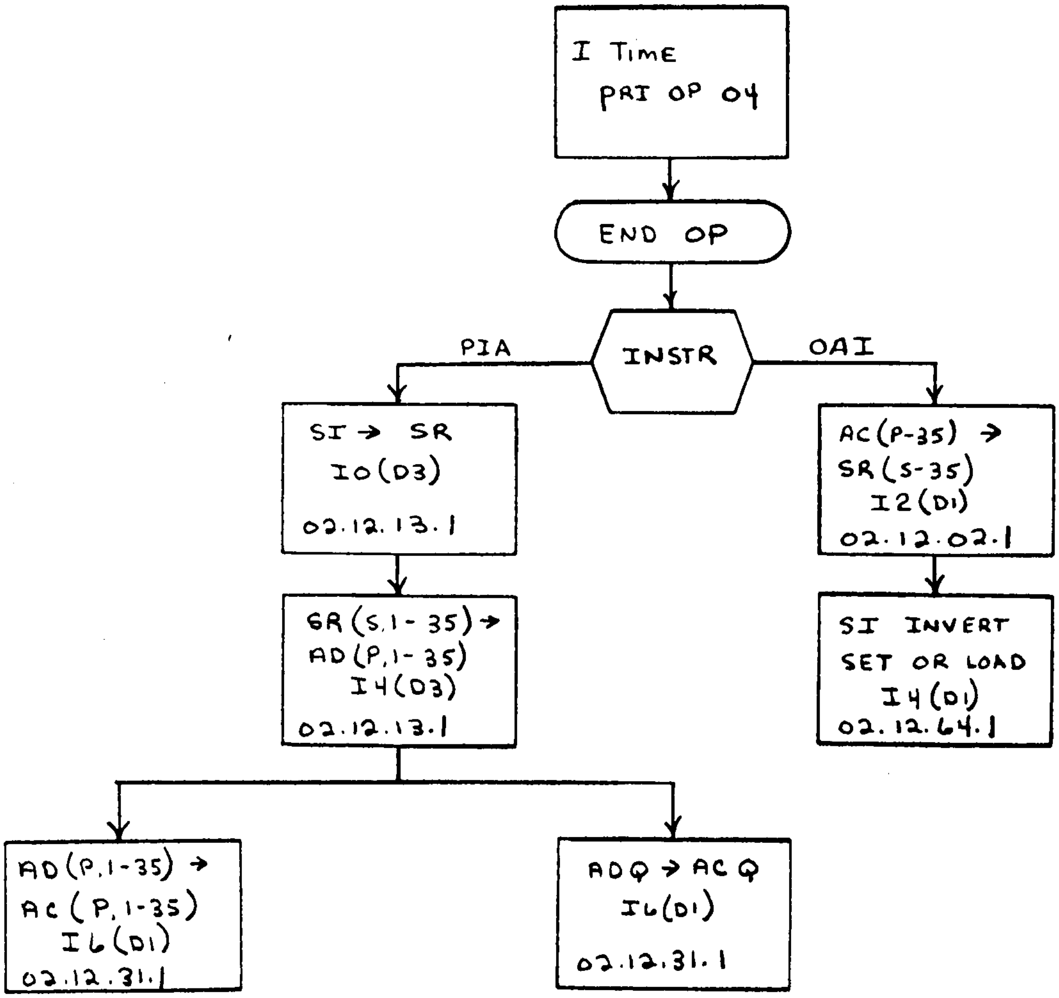


Figure 506-4B
page 500,34

OR Accumulator to Indicators OAI +0043 (I) Figure ~~5.3-55~~ 506-4

The logical OR of the contents of the accumulator (P, 1-35) and the indicators is placed in the sense indicator register. If a position in either register contains a one, a one will be placed in that position of the indicator. The accumulator is unchanged. Execution is accomplished using the sequence of PAI but omitting the indicator reset.

Reset Indicators from Accumulator RIA -0042 (I) Figure ~~5.3-55~~ 506-4

The positions of the sense indicator register which correspond to the positions of the AC(P, 1-35) having "1" bits are reset to zero. Indicator positions which correspond to "0" bits are unchanged. The AC is unchanged.

Invert Indicators from Accumulator IIA +0041 (I) Figure ~~5.3-55~~ 506-4

This instruction inverts any sense indicator position for which there is a corresponding "1" bit in the accumulator (P, 1-35).

Transfer if Indicators On TIO +0042 (I, L) DLA Figure ~~5.3-56~~ 506-5

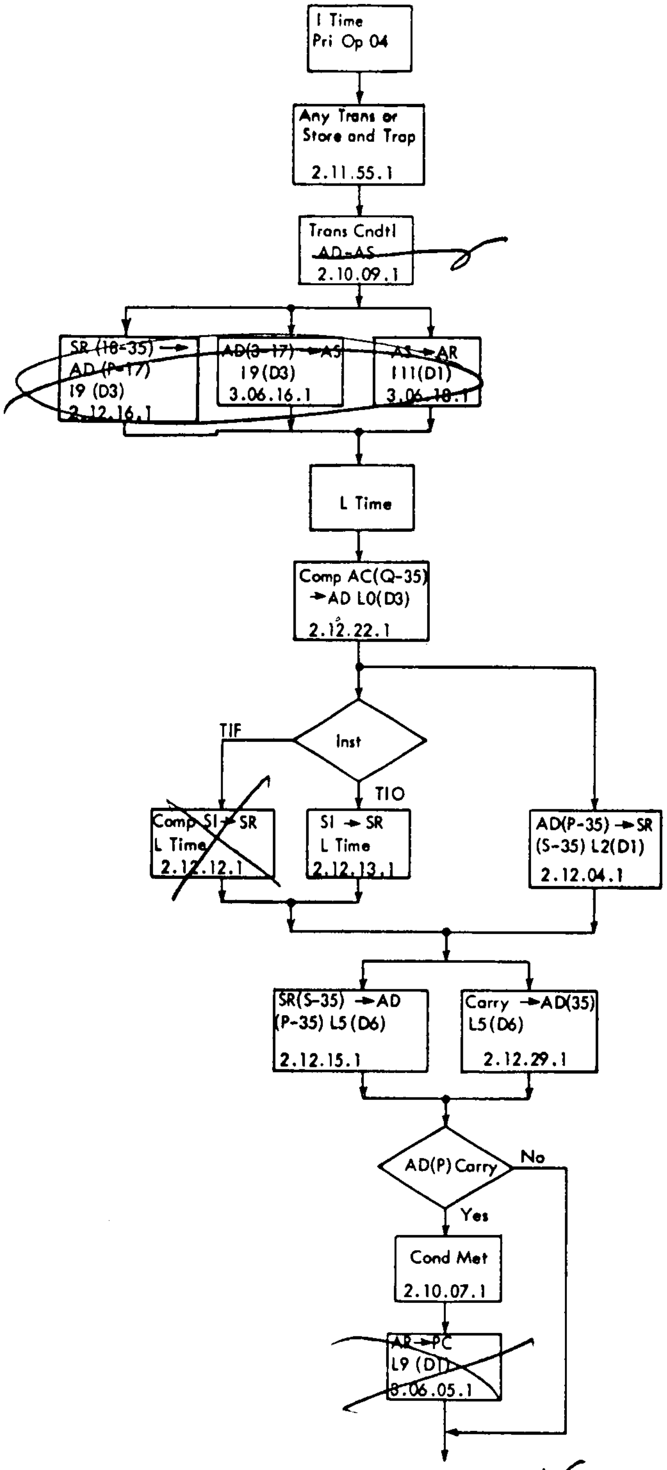
If all of the ones in the AC are matched by ones in the indicator register, an instruction transfer is taken to location X. AC positions containing zeros are not compared with indicator positions. If all of the ones are not matched, the computer takes the next instruction in sequence. To execute this instruction, the complement of the AC is OR'ed with the indicator. If the ones match, the result will be all ones. The result of the OR is stored in the SR and fed to the adder. A carry to adder (35) will ripple down the adder and carry out of the AD(P). The adder (P) carry is used to condition the transfer.

Transfer if Indicators Off TIF +0046 (I, L) DLA Figure ~~5.3-56~~ 506-6

If all of the ones in the AC are matched by zeros in the indicator register, an instruction transfer is taken to location X. AC positions containing zeros are not compared with indicator positions. If all of the ones are not matched, the computer takes the next instruction in sequence. Execution of this instruction is identical to that of TIO, except that the complement of the indicators is OR'ed to the complement of the AC. Test procedure remains the same.

On Test for Indicators ONT +0446 (I, E, 2L) DLA or TLA Figure ~~5.3-57~~ 506-6

If the ones contained in the word stored at location X are matched by ones in the corresponding indicator register positions, the next instruction will be skipped. If all of the ones are not matched, the computer will take the next instruction in sequence. Positions of storage location X which contain zeros are not compared. Execution of this instruction requires an E cycle to obtain the test word from storage and two L cycles to complete the test. The test is accomplished by moving the test word to the accumulator, where it can be complemented. The complement is OR'ed with the indicators and returned to the SR. The result of the OR will be all ones if the ones in the test word match the indicator. To test for all ones a carry is added to the OR which will result in an adder (P) carry to condition the advance instruction counter. The contents of the accumulator are saved and restored to normal during execution of the instruction.



506-5
FIGURE ~~5-56~~. TIO + 0042; ~~[illegible]~~

~~III~~
500.36

31

TIF - TRANSFER IF INDICATORS OFF +0046

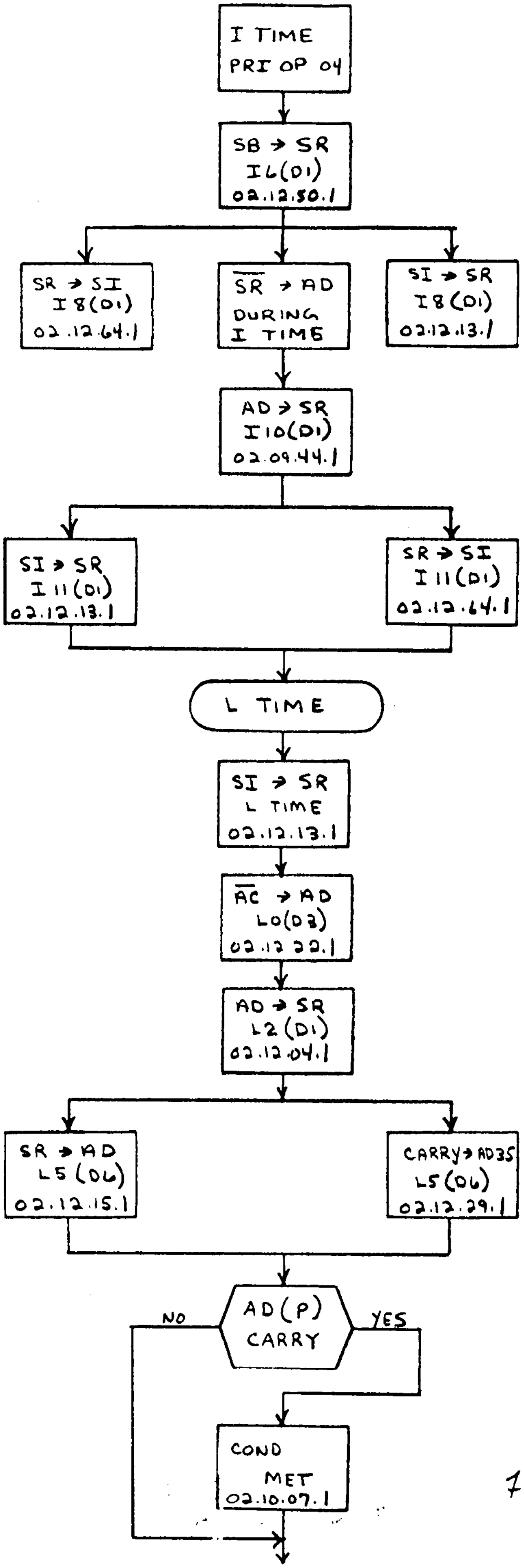


Figure 506-5A
page 500.36A

TIF (CON'T)

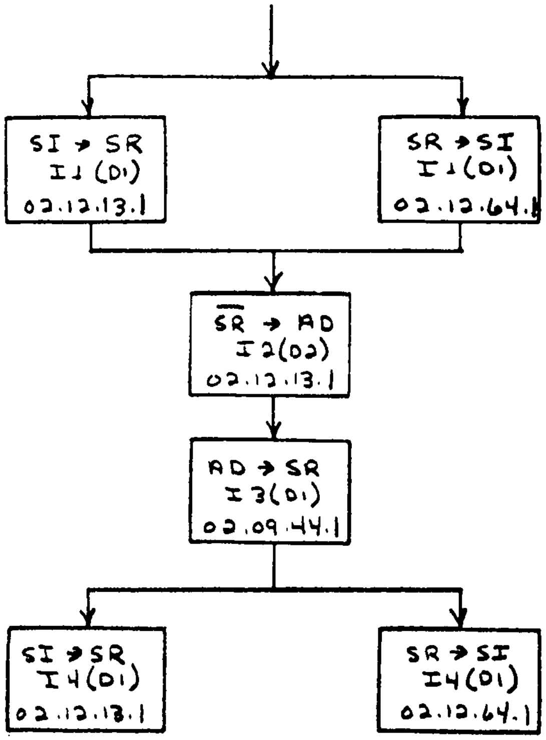


fig 506.5A (2)

page 500.36B

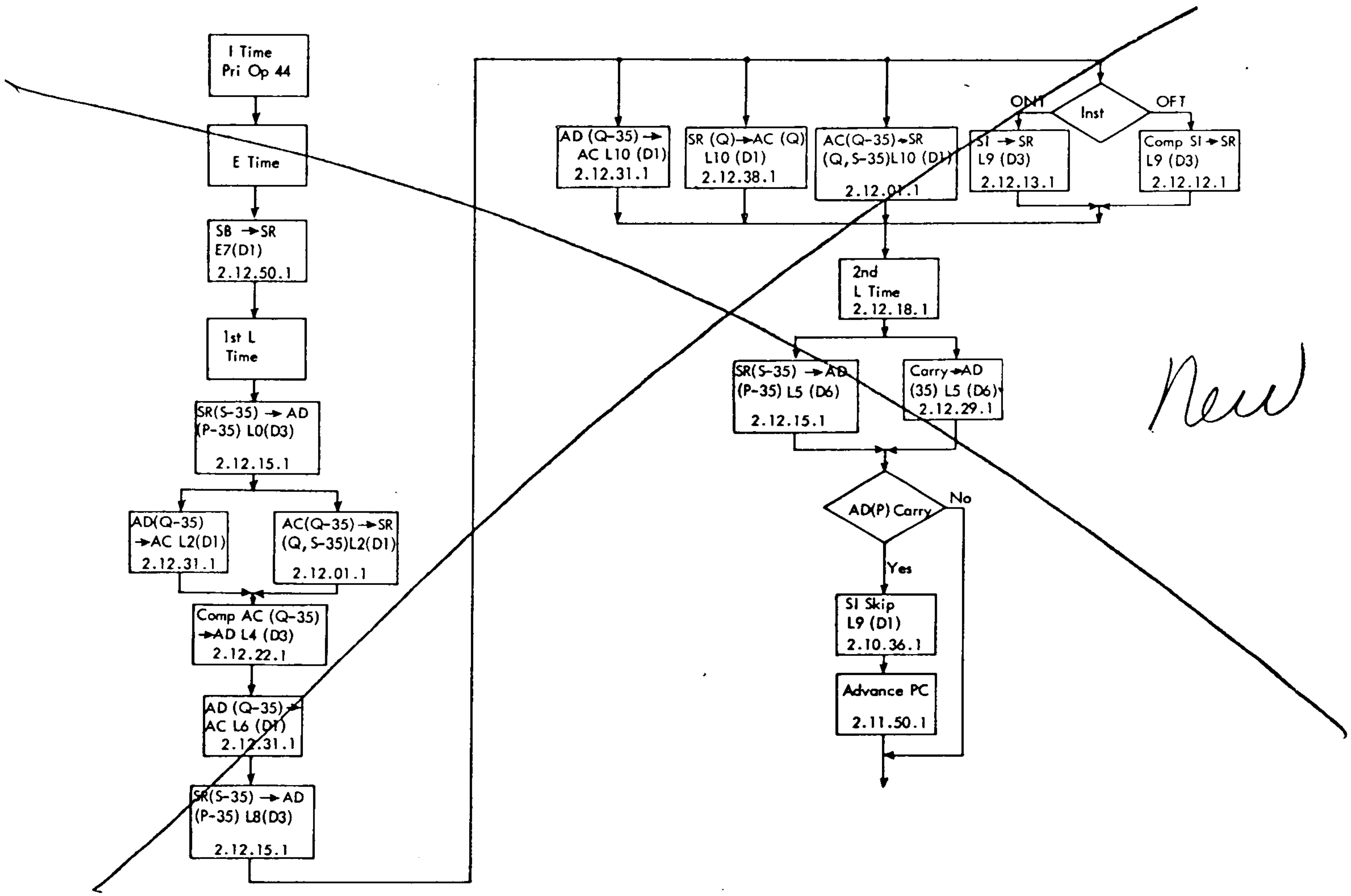


FIGURE 506-6. ONT +0446, OFT +0444

500.37

313
344

Off Test for Indicators OFT +0444 (I, E, L, L) Figure ~~5-3-57~~ 506-6
DLA or TLA

If the ones contained in the word stored at location X are matched by zeros in the corresponding indicator positions, the computer will skip one instruction. If all of the ones are not matched by zeros, the computer will take the next instruction in sequence. Positions in the test word from storage which contain zeros are not compared. Execution of this instruction is identical to that of ONT, except that the complement of the indicator is used for the OR operation instead of the true indicator.

Right-Half Indicators, On Test RNT +0056 (I, L, L) Figure ~~5-3-58~~ 506-7

This instruction matches the ones in the control field (18-35) of the instruction against the corresponding positions of the right half of the indicator register. If all of the ones are matched by ones, the computer skips one instruction. If all are not matched, the computer takes the next instruction in sequence. Positions of the control field containing zeros are not compared. Execution of this instruction is the same as ONT except that only positions (18-35) of SR are used.

Left-Half Indicators, On Test LNT -0056 (I, L, L) Figure ~~5-3-58~~ 506-7

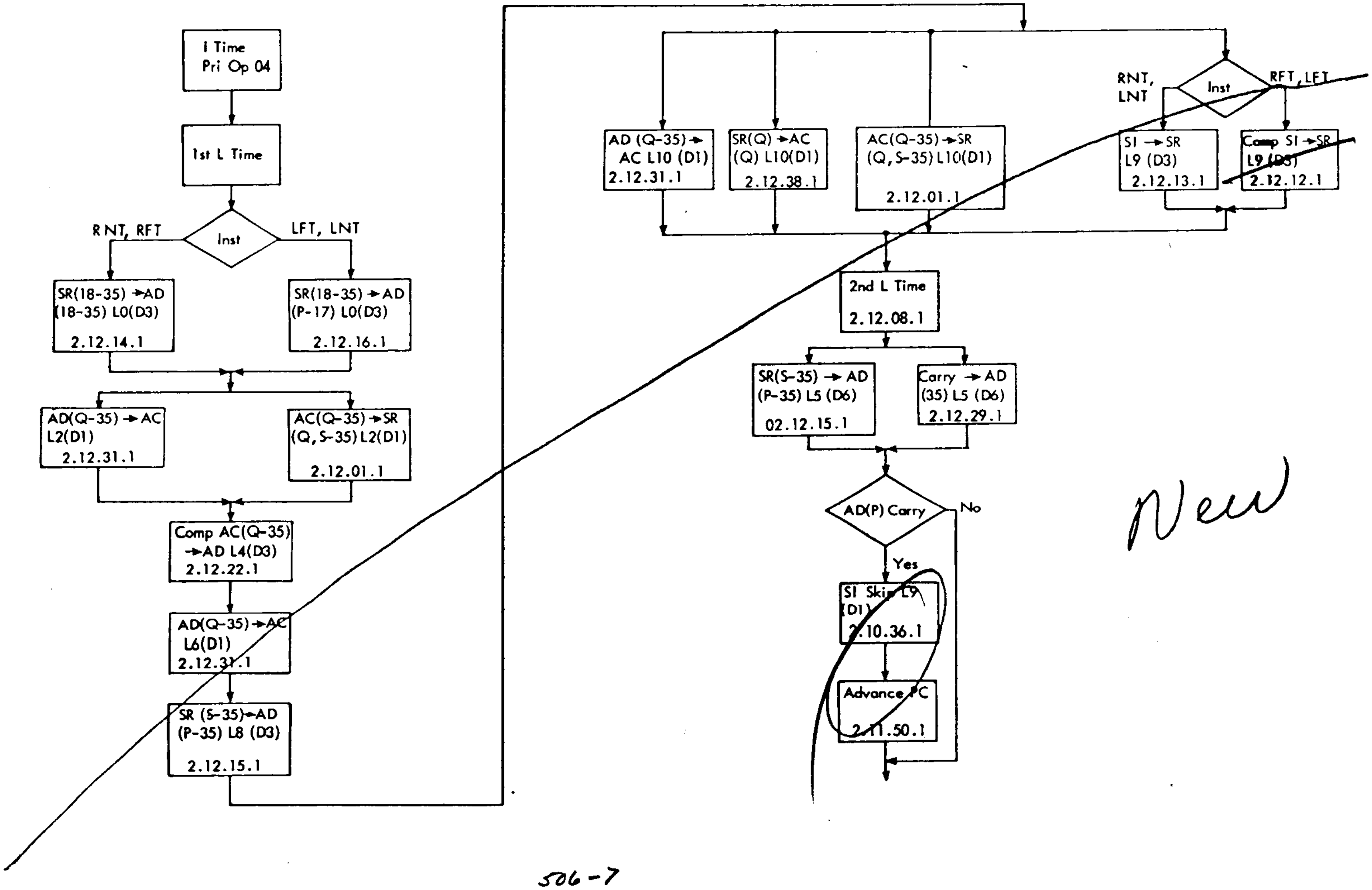
If the ones in the control field of the instruction are matched by ones in the corresponding positions of the left half of the indicator register, the computer will skip the next instruction. If all ones are not matched, the computer takes the next instruction in sequence. Execution of this instruction is the same as ONT, except that the SR (18-35) is compared against SI(0-17).

Right-Half Indicators, Off Test RFT +0054 (I, L, L) Figure ~~5-3-58~~ 506-7

If the ones in the control field of this instruction are matched by zeros in the right half of the indicator register, the computer will skip one instruction. If all ones are not matched by zeros, the computer will take the next instruction in sequence. Positions of the control field containing zeros are not compared. This instruction is executed the same as OFT, except that only positions (18-35) of the SR are used.

Left-Half Indicators, Off Test LFT -0054 (I, L, L) Figure ~~5-3-58~~ 506-7

If the ones in the control field of this instruction are matched by zeros in the left half of the indicator register, the computer will skip one instruction. If all are not matched, the computer will take the next instruction in sequence. Positions of the control field containing zeros are not compared. The control field and the indicator register are unchanged. Execution of this instruction is the same as OFT, except that SR(18-35) is compared with SI(0-17).


I Time
Pri Op 04
1st L Time
Inst
RNT, RFT
LFT, LNT
SR(18-35) → AD (18-35) L0(D3) 2.12.14.1
SR(18-35) → AD (P-17) L0(D3) 2.12.16.1
AD(Q-35) → AC L2(D1) 2.12.31.1
AC(Q-35) → SR (Q, S-35) L2(D1) 2.12.01.1
Comp AC(Q-35) → AD L4(D3) 2.12.22.1
AD(Q-35) → AC L6(D1) 2.12.31.1
SR (S-35) → AD (P-35) L8 (D3) 2.12.15.1
AD (Q-35) → AC L10 (D1) 2.12.31.1
SR(Q) → AC (Q) L10(D1) 2.12.38.1
AC(Q-35) → SR (Q, S-35) L10(D1) 2.12.01.1
Inst
RNT, LNT
RFT, LFT
SI → SR L9 (D3) 2.12.13.1
Comp SI → SR L9 (D3) 2.12.12.1
2nd L Time 2.12.08.1
SR(S-35) → AD (P-35) L5 (D6) 02.12.15.1
Carry → AD (35) L5 (D6) 2.12.29.1
AD(P) Carry
Yes
No
SI Skip L9 (D1) 2.10.36.1
Advance PC 2.11.50.1

New

FIGURE ~~5.2-8~~ 506-7. RNT +0056; LNT -0056; RFT +0054; LFT -0054

INDEX INSTRUCTIONS

The 7094 has seven 15-position index registers and can operate in one of two index modes. Multiple tag mode (set by the EMTM instruction) causes the computer to work on a 3 index register basis and provides compatibility with 7090 programs. Instruction positions 18, 19 and 20 indicate any one or any combination of the three index registers. Multiple bits in these positions cause OR'ing of the index registers and therefore defines the mode of operation.

Use of all seven index registers can be accomplished by leaving the multiple tag mode. (LMTM instruction). Instruction positions 18, 19 and 20 are decoded to indicate a specific index register; index OR'ing is not possible.

A 3-position tag register accepts and retains the tag bits from either the storage bus or I. B. R. depending on the various conditions of overlap. The output decoding of this register is dependent on whether the machine is in 3 or 7 index register mode.

A separate set of 15-position adders (index adders) have been placed in the 7094; they are similar in function but completely isolated from the normal 39-position main adder circuitry.

The output of the index registers is always gated to the index adders in 1's ~~address in 2's~~ complement form. The 1's complement output is converted to a 2's complement by a carry into index adder position 17. An indexable instruction may not be tagged; however, its address is always gated to the index adders and operated on by having an all 1's output from the index registers sent to the adders together with a carry to position 17. The net effect is to add 0's to the instruction address and no logic is performed.

Six index instructions are provided to test the various registers and transfer if the specified conditions are met. In addition, 16 instructions are provided to transfer data in true or complement form to or from the various index registers and the accumulator or core storage. In each case, either the address or decrement portion can be specified. Two instructions also load the index register in true or complement form from its own address positions.

A summary of these later index instructions is as follows:

A - To Index Register

1 - From core storage

	LXA	+0534	-	Load Index from Address
	LXD	-0534	-	Load Index from Decrement
	LAC	+0535	-	Load Index from Address Complemented
	LDC	-0535	-	Load Index from Decrement Complemented
	AXT	+0774	-	Address to Index True
	AXC	-0774	-	Address to Index Complemented

2 - From accumulator

	PAX	+0734	-	Place Address in Index
	PDX	-0734	-	Place Decrement in Index
	PAC	+0737	-	Place Address in Index Complemented
	PDC	-0737	-	Place Decrement in Index Complemented

B - From Index Register

1 - To storage

	SXA	+0634	-	Store Index in Address
	SXD	-0634	-	Store Index in Decrement
*	SCA	+0636	-	Store Index in Address Complemented
*	SCD	-0636	-	Store Index in Decrement Complemented

2 - To accumulator

	PXA	+0754	-	Place Index in Address
	PXD	-0754	-	Place Index in Decrement
*	PCA	+0756	-	Place Index in Address Complemented
*	PCD	-0756	-	Place Index in Decrement

* New 7094 Instructions

Load Index From Address - LXA +0534 ~~+0543~~ I, E TLA (Figure 507-1)

This instruction makes reference to a core storage location and loads the specified index register with the contents of positions 21-35.

At I7 time, the storage bus is gated into the storage register and positions 21-35 routed immediately into the address register. At the beginning of the next E cycle, this address is sent to MAR. Address modification is not possible; gating circuitry which normally takes the address register to the index adders and back again is blocked. Decoding from the tag register gates index registers' outputs into the index adders together with a carry to position 17; no logic is performed, however, because the index adder outputs are not gated back through any circuitry.

During the E cycle, the storage bus is set into the storage register at 7-time and positions 21-35 are routed to the address register at 11-time. At the beginning of the next I cycle, the address register contents are routed to the index adders by an IO (D3) pulse and from there, set into the index register by an I2 CP set pulse. The set pulse resets the index register to clear out old information; the same pulse, delayed by circuitry (03.05.33.1), then overides the reset pulse and causes the index register to be set to the new value.

At the same time that the IO (D3) pulse is gating the address register to the index adders, an A2 (D2) pulse is also bringing up "gate AR - XAD" on 03.06.06.1. This line however, performs no logic in this operation.

<u>Load Index From Decrement</u> - LXD -0534 I-E TLA (Figure 507-1)

This instruction makes reference to a core storage location and loads the specified index register with the contents of positions 3-17. The initial phase of the operation is similar to LXA; positions 21-35 of the LXD instruction are gated to the address register and out to MAR for the E cycle core storage reference. Address modification is blocked as previously explained in LXA.

During the E cycle, the storage bus is set into the storage register at 7-time and positions 21-35 routed to the address register at 11-time. The routing to the address register at this time, however, performs no logic during this operation.

At the beginning of the next I cycle, storage register positions 3-17 (decrement) are routed directly to the index adders by an IO (D3) pulse; circuitry from the address register to the index adders is blocked. An I2 CP set pulse resets the selected index register and samples in the new value from the index adders to complete the operation.

Load Index From Address Complemented - LAC +0535 I, E TLA (Figure 507-1) This instruction makes reference to a core storage location and loads the specified index register with the complemented contents of positions 21-35.

The initial phase of the operation is similar to LXA; positions 21-35 of the LAC instruction are gated to the address register and out to MAR for the E cycle core storage reference. Address modification is blocked as previously explained in LXA.

During the E cycle, the storage bus is set into the storage register at 7-time and positions 21-35 are routed to the address register at 11-time. At the beginning of the next I-cycle, the address register contents are routed to the index adders by an AO (D3) pulse and from there, set into the index register by an I2 CP set pulse. This pulse accomplishes both the resetting and setting of the selected index register.

At the same time that the IØ (D3) pulse is gating the address register to the index adders, an A2 (D2) pulse is also bringing up "gate AR - XAD" on 03.06.06.1. This line, however, performs no logic in the operation.

At I2 time the index register contains the true value. In order to replace this with the 2's complement, the index register (which always comes out in 1's complement form) is returned to the index adders with a carry to position 17 at I4 (D2). An I5 CP set pulse places the complemented value back into the index register and the operation is completed.

Load Index From Decrement Complemented - LDC -0535 I, E TLA (Figure 507-1)

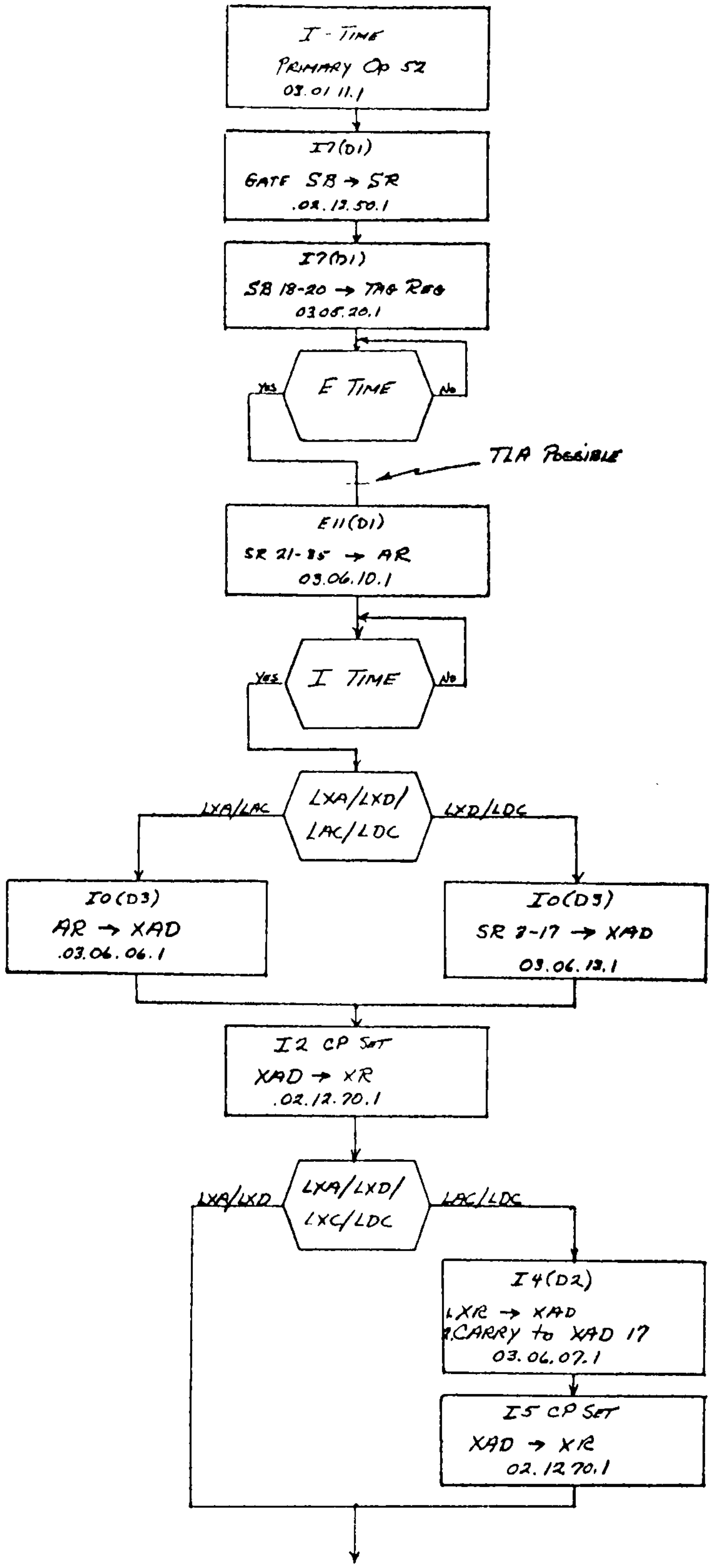
This instruction makes reference to a core storage location and loads the specified index register with the complemented contents of positions 3-17.

The initial phase of the operation is similar to LXA; positions 21-35 of the LDC instruction are gated to the address register and out to MAR for the E cycle core storage reference. Address modification is blocked as previously explained in LXA.

During the E cycle, the storage bus is set into the storage register at 7-time and positions 21-35 routed to the address register at 11-time. The routing to the address register at this time, however, performs no logic during this operation.

At the beginning of the next I cycle, storage register positions 3-17 (decrement) are routed directly to the index adders by an I0(D3) pulse; circuitry from the address register to the index adders is blocked. An I2 CP set pulse resets the selected index register and samples in the new value from the index adders.

At I2 time, the index register contains the true value. In order to replace this with the 2's complement, the index register (which always comes out in 1's complement form) is returned to the index adders with a carry to position 17 at I4(D2). An I5 CP set pulse places the complemented value back into the index register and the operation is completed.



LXA/LXD/LAC/LDC FLOW CHART

FIGURE 507-1

Address to Index True - AXT +0774 I (no overlap) Figure 507-2

This instruction loads positions 21-35 of the AXT into the specified index register in true form.

At I7 time, the storage bus is gated into the storage register and positions 21-35 immediately routed into the address register. Address modification is not possible (due to bits in PR 6 and 7); no gating is generated to take the address register to the index adders and back again (03.06.06.1). Decoding from the tag register, however, gates the index register to the index adders with a carry to position 17. No logic is performed.

AXT/AXC decoding forces an "end opn XR cntls" (02.15.64.1). The incremented program counter is sent to MAR and the computer takes a next I time. During the initial portion of this cycle an I0 (D3) pulse gates the address register to the index adders and from there into the index register at I~~1~~2 CP set pulse time. An A2 (D2) pulse (03.06.06.1) is also gating the address register to the index adders, producing an OR'ing condition with the I0 (D3).

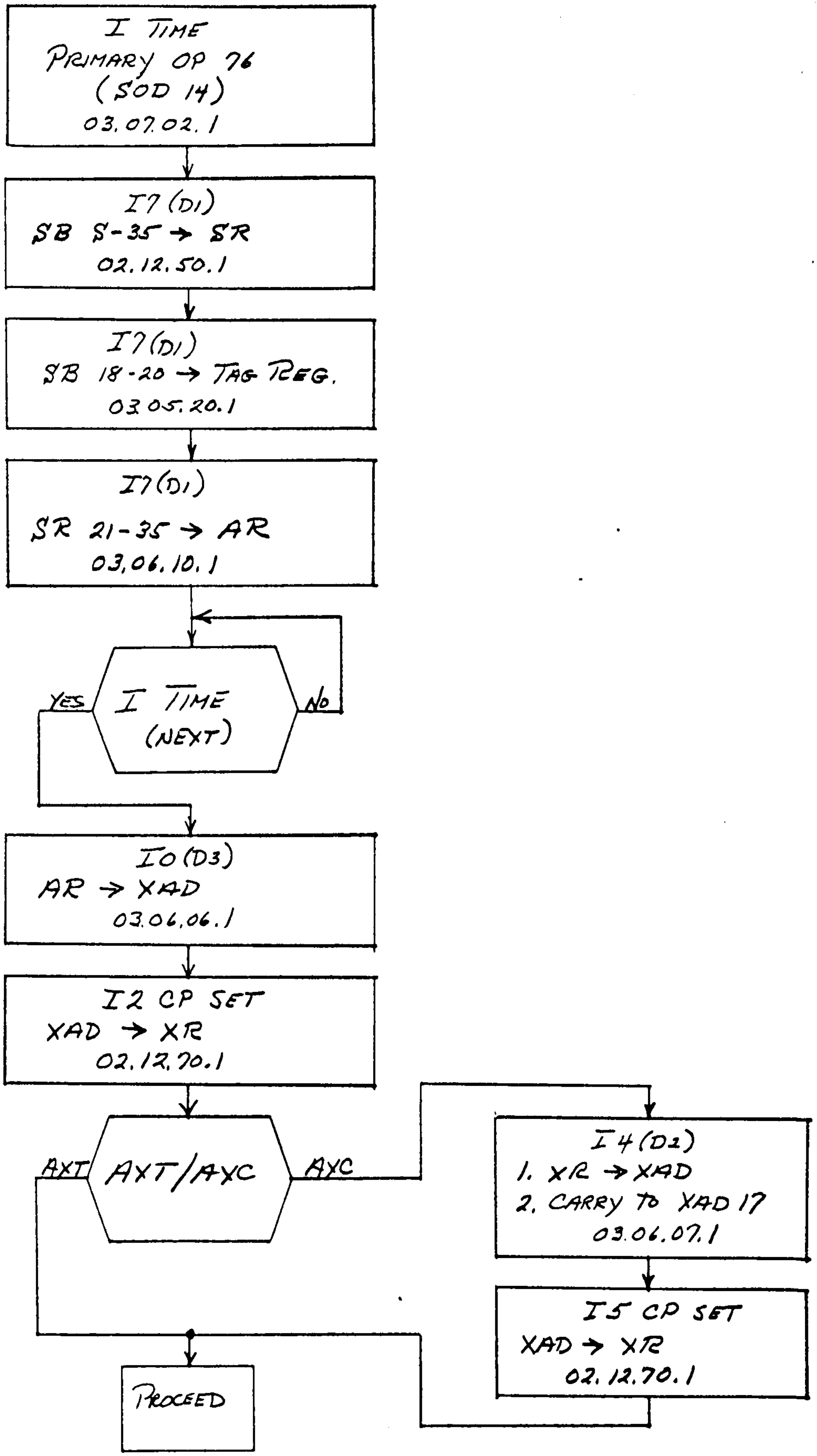
As this is a primary Op 76 instruction, the index adders are unconditionally gated into the shift counter at I10 time. The value in the shift counter will be the 2's complement of the indicated index register.

Address to Index Complemented - AXC -0774 I (no overlap) Figure 507-2

This instruction loads positions 21-35 of the AXC into the specified index register in 2's complement form.

The first I cycle is identical to AXT -- the storage bus is gated into the storage register and positions 21-35 routed to the address register at I7 time. The incremented program counter is gated to MAR and the instruction ends operation and proceeds to the next I cycle. Address modification is not possible.

During the next I cycle, an I0 (D3) pulse gates the address register to the index adders. An I2 CP set pulse places the value into the index register in true form. Complementing is accomplished by routing the index register (which always comes out in 1's complement form) to the index adders with a carry to position 17 at I4 (D2) time . An I5 CP set pulse gates the complemented value back into the index register and the operation is complete.



AXT/AXC FLOW CHART

Figure 507-2

PLACE ADDRESS IN INDEX - PAX +0734 I No Overlap - (Figure 507-6)

These next four instructions are concerned with placing information from the accumulator into specified index registers. They require only one (I) cycle and, therefore, cannot be overlapped by an instruction in a next higher odd address.

The object of PAX is to place positions 21-35 of the accumulator (the address portion) into the specified index register. At ~~I7 time~~ I6(D2) and I7(D1) respectively the storage register and tag register are set from the storage bus, and SR positions 21-35 immediately routed to the address register. Setting the storage register and gating 21-35 to the AR has no logic at this time but functions because of normal I-time circuitry.

Address modification is not possible; gating circuitry which normally takes the address register to the index adders and back again is blocked. Tag register decoding, however, does gate the specified index register to the index adders along with a carry to XAD 17. This occurs as a normal I-time function and performs no logic because the XAD outputs are not gated back again.

The accumulator contents must be placed in the storage register in order to obtain routing paths for data-flow to the index register. To accomplish this, the accumulator is routed to the storage register at I10 time; at I11 time SR positions 21-35 are routed to the address register. Thes e preliminary routings and functions during the first I cycle are common to all four POD 72 instructions (PAX, PDX, PAC, PDC).

During the next (I) cycle, the address register contents are routed through the index adders by an I0 (D3) pulse and set into the specified index register at I2 CP set time to complete the operation.

At the same time that the I0 (D3) pulse is gating the address register to the index adders, an A2 (D2) pulse is also bringing up "gate AR-XAD" on 03.06.06.1. This produces an OR'ing condition to the I0 (D3) but performs no logic in the operation.

326

PLACE DECREMENT IN INDEX - PDX -0734 I   No Overlap - (Figure 507-6)

The PDX instruction places positions 3-17 of the accumulator (the decrement portion) into the specified index register. All of the initial I-time functions occur as explained for PAX; the accumulator contents are routed to the storage register at I10 time. Positions 21-35 of the SR are routed to the address register as a normal POD 72 function but performs no logic during the PDX instruction.

At I0 (D3) time of the next (I) cycle, positions 3-17 of the storage register are routed directly to the index adders and set in the specified index register at I2 CP set pulse time. I0 (D3), "gate AR-XAD", circuitry (03. 06. 06. 1) is blocked by a PDX condition.

321

PLACE ADDRESS IN INDEX COMPLEMENTED - PAC +0737 I   No Overlap

(Figure 507-6)

The PAC instruction places the complemented contents of accumulator positions 21-35 (address portion) into the specified index register. All of the initial I-time functions occur as explained for PAX. The accumulator contents are routed to the storage register at I10 time and SR 21-35, then, immediately routed to the address register at I11 time.

During the initial portion of the next (I) cycle, the address register contents are routed through the index adders by an I0 (D3) pulse and set into the specified index register at I2 CP set time. This places a true value in the index register.

Complementing is accomplished by routing the index register (which always comes out in 1's complement form) to the index adders with a carry to XAD 17 at I4 (D2) time. An I5 CP set pulse gates the complemented value back into the index register to complete the operation.

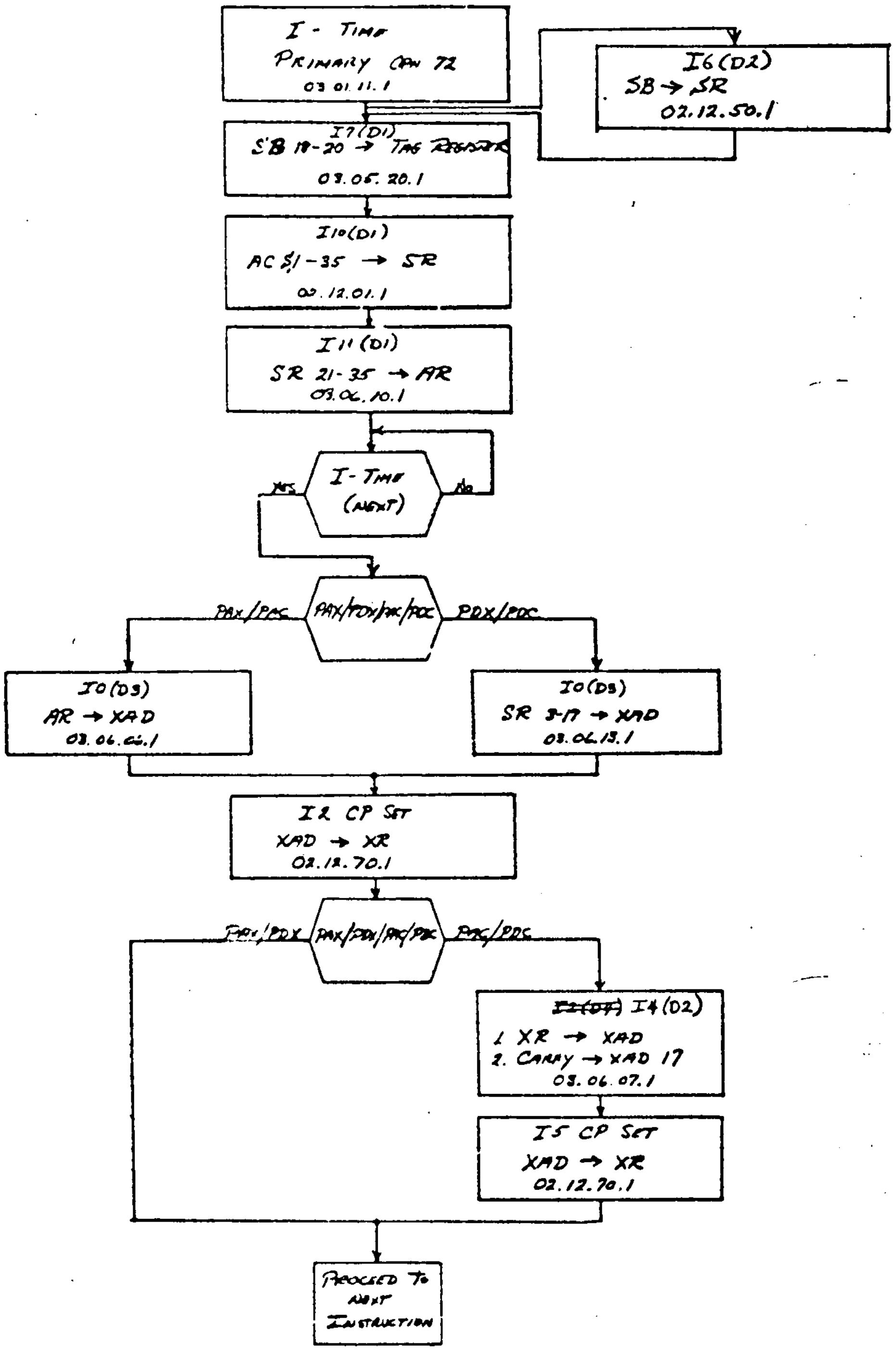
PLACE DECREMENT IN INDEX COMPLEMENTED - PDC -0737 I No Overlap

(Figure 507-6)

The P~~A~~DC instruction places the complemented contents of accumulator positions 3-17 (decrement portion) into the specified index register. All of the initial I-time functions occur as explained for PAX; the accumulator contents are routed to the storage register at I10 time. At I11 time, SR positions 21-35 are routed to the address register as a normal POD 72 function but performs no logic during the PDC instruction.

At I0 (D3) time of the next (I) cycle, positions 3-17 of the storage register are routed directly to the index adders and set into the specified index register at I2 CP set pulse time. This places a true value in the index register.

Complementing is accomplished by routing the index register (which always comes out in 1's complement form) to the index adders with a carry to XAD 17 at I4 (D2) time. An I5 CP set pulse gates the complemented value back to the index register to complete the operation.



PAX/PDX/PAC/PDC FLOW CHART

FIGURE 507-6

STORE INDEX IN ADDRESS - SXA + 0634 I, E NO OVERLAP (FIGURE ~~8~~ 500.50.11)

These next four instructions are concerned with taking information from the specified index register and storing it in true or complemented form in either the address or decrement portion of the indicated core storage location. Two cycles are required (I and E) for their execution and overlapping is not permitted by an instruction in the next higher odd location.

The object of the SXA is to store the true value of the index register into the address portion of the specified core storage location. At ~~17 time~~ I6(D2) and I7(D1) time respectively, the storage register and tag register are set from the storage bus and SR positions 21-35 immediatel ely routed to the address register. At the beginning of the next E cycle, this value is sent to MAR for the E cycle core storage reference. Address modification is not possible; gating circuitry is blocked which normally takes the AR to the index adders.

Also, during the first I time, tag register decoding gates the specified index register contents to the index adders with a carry to XAD 17. This occurs at I9 (D2) as a normal I-time function; however, no logic is performed during the SXA instruction because the adders are not gated beyond this point.

The index register contents must be routed to the storage register in order to obtain a data-flow path to the storage bus and core storage. A routing path is available from the index adders to the storage register but it must be remembered that this path has the capability of routing only the complement of the index adders.

During the early portion of the E cycle at E0 (D3) time, the index register is again gated to the index adders. Because the contents of the index registers are always brought out in 1's complement form, the adders now contain the 1's complement of the value desired. An E1 (D1) pulse gates the complement of the index adders to positions 21-35 of the storage register; the true index register value is now in the SR and ready to be sent to core storage via the storage bus. "Store Address" controls are active because of SXA operation decoding, and the address portion of core storage is modified without disturbing the remainder of that data word.

Multiple tagging is possible with this group of instructions. In # order to maintain compatibility with previous systems, the specified index registers must be ORed together. Therefore, (for the SXA instruction) at the same time that the index adders are being gated to the SR, the ORed output is set into the specified index registers. The index registers now contain the 1's complement of the ORed values. (If multiple tagging is not specified, the index register contains the 1's complement of the original value.) The index register (or registers) is returned to a true value by gating the output to the index adders and back again during the following E4(D2) time.

500.50.11

STORE INDEX IN DECREMENT - SXD -0634 I, E NO OVERLAP (FIGURE 19)

The SXD instruction stores the true value of the index register into the decrement portion of a specified core storage location. All of the initial I-time functions occur as explained for SXA. Positions 21-35 of the storage register are routed to the address register for core storage reference during the next E cycle.

During the early portion of the next E cycle at E0 (D3) time, the index register contents (which always come out in 1's complement form) are gated to the index adders where the complement of the index adders is routed and set into positions 21-35 of the storage register. Double complementing produces the true index register value in the SR.

The ORed complement of the index register (or registers) also replaces the original contents at E1 time; the true ORed value is restored during the following E4(D2) time.

Positions 21-35 of the storage register must be moved into positions 3-17 in order to accomplish storing the desired value in the decrement of a core storage location. This swapping of the address and decrement is accomplished by routing 18-35 of the SR to P-17 of the adders; the adders are then routed back to and set into the storage register. Positions 18-35 of the storage register are reset. The storage register is gated to core storage to complete the operation.

"Store decrement" controls are active because of SXD operation decoding, and the decrement portion of core storage is modified without disturbing the remainder of the data word.

STORE COMPLEMENT OF INDEX IN ADDRESS - SCA +0636 I, E NO OVERLAP (FIGURE pg 500.50.1)

The SCA instruction stores the 2's complement of the index register value into the address portion of a specified core storage location. All of the initial I-time functions occur as explained for SXA but with one addition; the 2's complement is gated from the index adders and set into the index register at I10 time. The address register contains the core storage reference for the next E cycle.

During the early portion of the next E cycle at E0 (D.3) time the index register contents are gated to the index adders. The index register contains the 2's complement of the original value; the output, however, is the 1's complement of that. The complement of the index adders is routed and set into positions 21-35 of the storage register. This double complementing effectively places the 2's complement of the original index register contents into the SR where it is placed on the storage bus and sent to core storage. "Store Address" controls allow modification of the address portion without disturbing the remainder of the data word.

At this point the index register contains a complemented value. Restoring the original contents is accomplished during the next I-time when the index register (or registers) is gated to the index adders with a carry to XAD 17. Taking this output back to the index register completes the operation.

STORE COMPLEMENT OF INDEX IN DECREMENT -0636 I, E NO OVERLAP (FIGURE 500.50)

The SCD instruction stores the 2's complement of the index register value into the address portion of a specified core storage location. All of the initial I-time functions occur as explained for SXA but with one exception; the 2's complement is gated from the index adders and set into the index register at I 10 time. The address register contains the core storage reference for the next E cycle.

During the early portion of the next E cycle at E0 (D3) time the index register contents are gated to the index adders. The index register contains the 2's complement of the original value; the output, however, is the 1's complement of that. The complement of the index adders is routed and set into positions 21-35 of the storage register. This double complementing effectively places the 2's complement of the original index register contents into the SR. Positions 21-35 of the storage register must be moved into positions 3-17 in order to accomplish storing the desired value in the decrement of a core storage location. This swapping of the address and decrement is accomplished by routing 18-35 of the SR TO P-17 of the adders; the adders are then routed back to and set into the storage register, where the contents are placed on the storage bus and set into the MDR. Positions 18-35 of the storage register are reset. "Store decrement" controls are active because of the SCD operation decoding and the decrement portion of core storage is modified without distrubing the remainder of the data word.

At this point the index register contains the complemented value. Restoring the original contents is accomplished during the next I time when the index register is gated to the index adders with a carry to XAD 17. Taking this value back to the index register completes the operation.

PLACE INDEX IN ADDRESS   PXA   +0754   I   No Overlap

(Figure 500.51.5)

These next four instructions are concerned with taking information from the specified index register and placing it in true or 2's complement form in either the address or decrement portion of the accumulator. These are one (I) cycle instructions; therefore, overlapping is not permitted by an instruction in the next higher odd core storage location.

The object of the PXA instruction is to place the true value of the index register into the address portion of the accumulator. The data flow path to the accumulator is: from the index register, through the index adders, to the storage register, and through the main adders into the accumulator.

At ~~I7 time,~~ I6(D2) and I7(D1) time respectively, the storage register and tag register are set from the storage bus; SR positions 21-35 are immediately routed to the address register as a normal I-time function, but performs no logic in this operation.

During the first I-time, the tagc register decoding gates the specified index register contents to the index adders with a carry to XAD I7 at I9 (D2) as a normal I-time function; however, no logic is performed during the PXA instruction because the adders are not gated beyond this point. The normal I9 (D2) gating of the address register to the index adders at this time is belocked because this is a non-indexible instruction.

During the early portion of the next (I) cycle at I0 (D3) time, the index register is again gated to the index adders. Because the contents of the index registers are always brought out in 1's complement form, the adders now contain the 1's complement of the value desired. An I1 (D1) pulse gates the complement of the index adders to

500.51.0

positions 21-35 of the storage register; the true index register value is now in the storage register and ready to be sent to the accumulator via the main adders.

Multiple tagging is possible with this group of instructions. In order to maintain compatibil ity with previous systems, the specified index registers must be ORed together. Therefore, (for the PXA instruction) at the same time that the index adders are being gated the SR, the ORed output is set into the specified index registers. The index registers now contain the 1's complement of the ORed values . If multiple tagging is not specified, the index register contains the 1's complement of the original value. The index register (or registers) is returned to a true value by gating the output to the index adders and back again during I4 (D2) time.

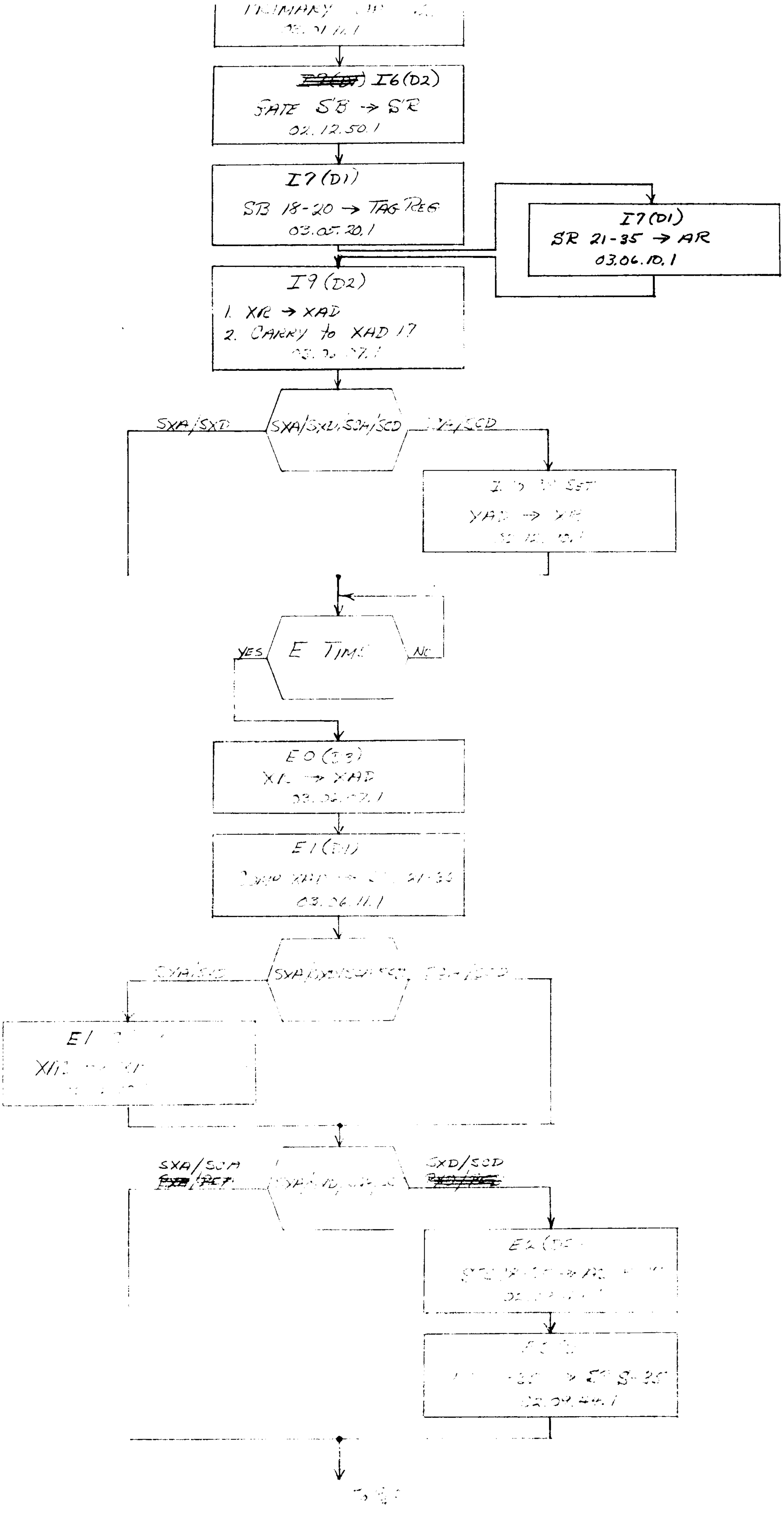
With the index register value in the storage register, the operation is completed by routing the SR through the main adders and setting it into the accumulator during I4 (D3) time. The accumulator is cleared prior to the setting; therefore, if no tag is specified, the accumulator contains all 0's, at the end of the operation.

Place Index in Decrement - PXD -0754 I No Overlap (Figure

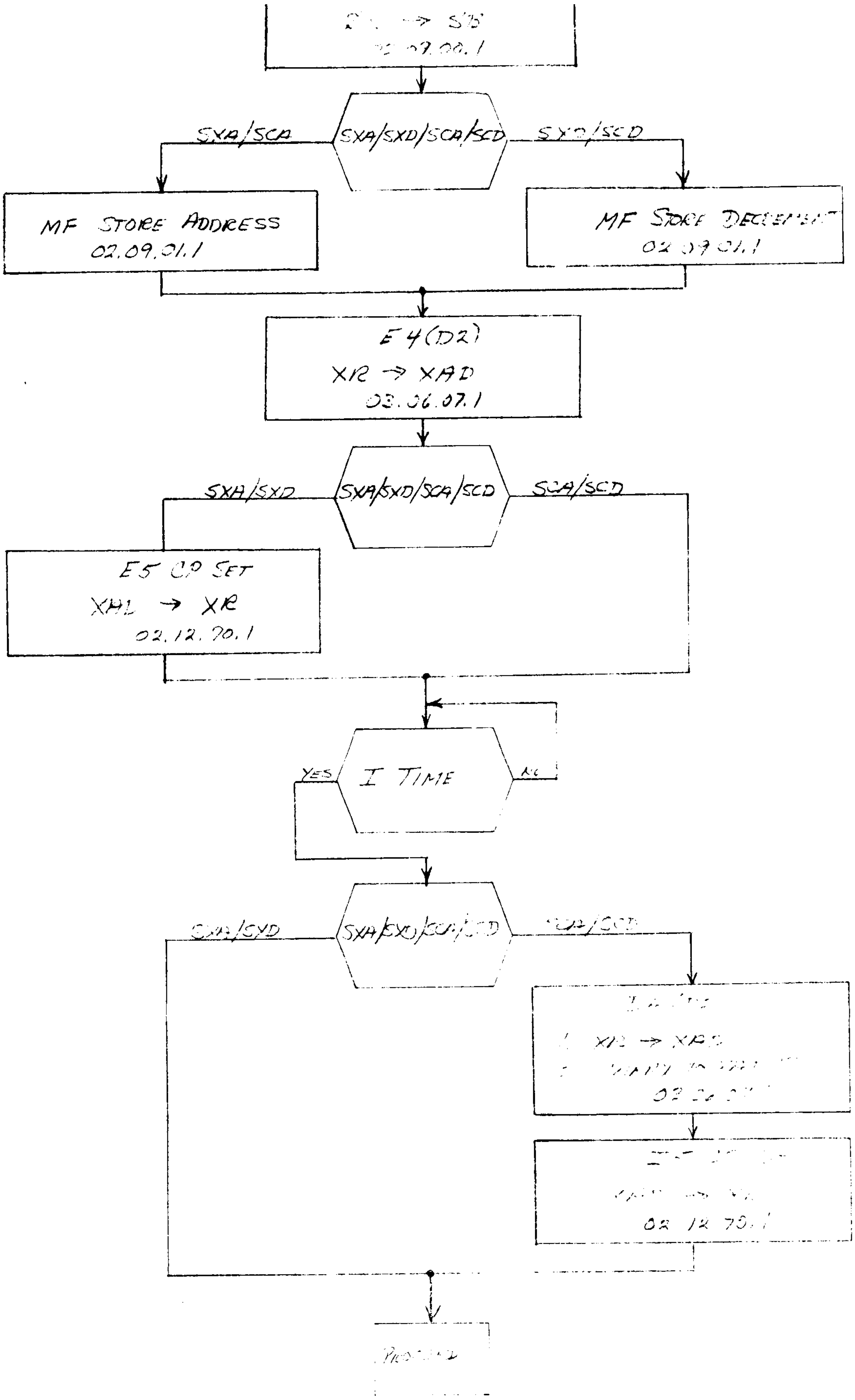
The PXD instruction places the true value of the index register into the decrement portion of the accumulator. The data flow path is similar to the PXA instruction; the initial I-time functions are identical.

During the early portion of the next (I) cycle, the index register is again routed to the index adders and from there to positions 21-35 of the storage register. The OR'ed 1's complement of the index registers (in the case of multiple tagging) also replaces their original contents at I1 (D1) time. The true value is restored again during the following I4 (D2) time.

With the index register value in the storage register, the operation is completed by routing positions 18-35 to positions P-17 of the adders and from there into the decrement portion of the accumulator. The remaining positions of the accumulator are cleared.



SXA/SXD/SCA/SCD FLOW CHART



SXA/SXD, SCA/SCD Flow Chart

PLACE COMPLEMENT OF INDEX IN ADDRESS PCA +0756 I

No Overlap (Figure p8 500.51.5)

The PCA instruction places the 2's complement of the index register into the address portion of the accumulator. The data-flow path is identical to PXA; the initial I-time functions are identical with one exception. When the contents of the index register (or registers) is gated out at I9 (D2) time with a carry to XAD I7, an I 10 CP set pulse sets this 2's complement value back into the index register. This is necessary so that the complemented value will be available to be sent to the storage register.

During the early portion of the next (I) cycle the index register is again routed to the index adders and from there to the storage register. The 1's complement of the index register coming into the adders and the complement of the index adders being sent to the storage register effectively places the present index register value in the SR. The present value, however, is the ~~2nd~~ 2's complement of the original index register contents. The true value is restored again during the following I4 (D2) time when the index register and a carry are routed to the index adders and back again.

With the 2's complement index register value in the storage register, the operation is completed by routing the contents of the SR through the main adders and into the accumulator.

PLACE COMPLEMENT OF INDEX IN DECREMENT PCD -0756 I

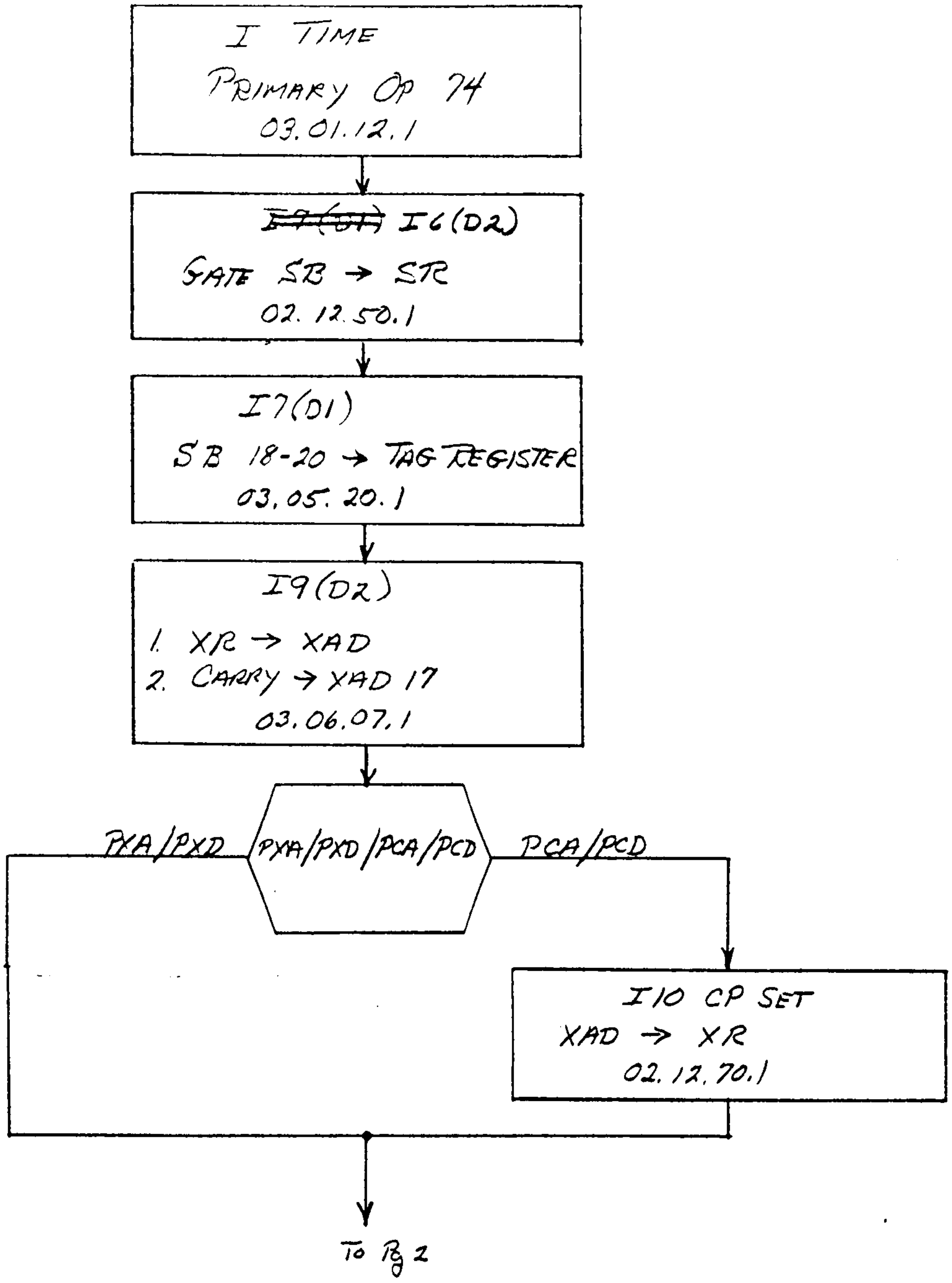
No Overlap (Figure ~~8~~ 500.51.5)

The PCD instruction places the 2's complement of the index register into the decrement portion of the accumulator. The data-flow path is similar to the PXA instruction; initial I-time functions are also the same except that the 2's complement is generated and set into the index register during I9 (D2) time.

During I0 (D3) time of the next cycle, the index register is routed through the index adders and set into positions 21-35 of the storage register. The storage register now contains the 2'd complement of the original index register contents. The true index register value is restored again during the following I4 (D2) time when the index register and a carry are routed to the index adders and back again to the index register.

With the 2's complement index register value in the storage register, the operation is completed by routing positions 18-35 to positions P-17 of the main adders and from there into the decrement portion of the accumulator.

500,51,4



PXA/PXD/PCA/PCD FLOW CHART

FIGURE ____________

500.51.5

Pg 1 of 2

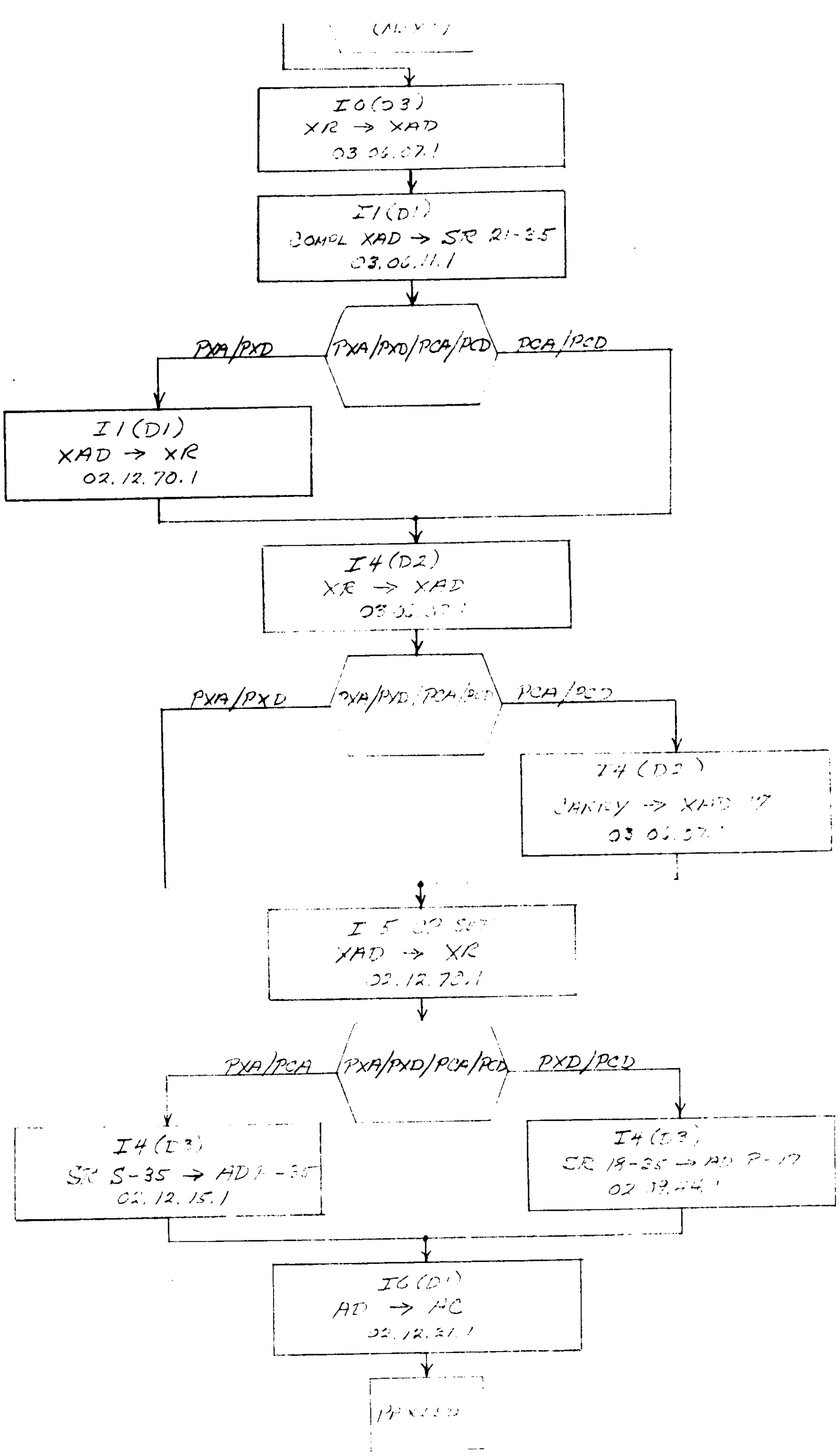
B. 7/62

TRANSFER ON INDEX TIX +2000 I No Overlap (Figure 98 500.57)

These next four instructions are concerned with comparing their decrement portion with the contents of the specified index register. High, equal, or low indications cause the computer to either proceed to the next sequential instruction or transfer to the core storage location specified in the address portion of the instruction being executed. These are one (I) cycle instructions; therefore, overlapping is not permitted by an instruction in the next higher odd core storage location. Bit recognition in positions 1 or 2 of the storage bus (02.11.40.1) blocks sending positions 3 - 11 to the program register; instead, positions 1 and 2 are gated to PR positions 8 and 9. Decoding at this point recognizes these as non-indexable instructions (02.12.76.1).

The TIX instruction compares its decrement with the contents of the specified index register. If the number in the specified index register is greater than the decrement, the contents of the index register are reduced by the amount of the decrement and the computer transfers to location Y. When the number in the decrement is equal to or less than the decrement, no reduction is made and the computer takes the next instruction in sequence.

At I6(D2) and I7(D1) time respectively, ~~I7 time~~, the storage register and tag register are set from the storage bus; SR positions 21-35 are immediately sent to the address register to provide a transfer address if the decrement value is not greater than the index register.



PXA/PXD/PCA/PCD FLOW CHART

Figure ______________

At I8(D2) time delayed, ~~I9(D2) time~~ positions 3 - 17 of the storage register are routed to the index adders. At the same time, the index register and a carry to XAD 17 are also gated to the adders. (Because this is a non-indexable instruction, circuitry is blocked that would normally take the address register to the index adders.)

A ripple carry out of XAD3 sets an XAD3 carry trigger and indicates that the decrement value is equal to or greater than the index register. Under these conditions the program counter is gated to MAR and the computer continues with the next sequential instruction. No carry from XAD3 indicates that a successful reduction is possible, and the adders are gated back to the index register. The result from the adders is the 2's complement of the true value; this is corrected during the next I4(D2) time when the index register and a carry to XAD 17 are gated to the index adders and back again.

With no XAD3 carry, a "set condition met" trigger is turned ON which causes address register gating to MAR, and the computer proceeds at the transfer address. During the following I2(D2) time the address register is gated to the index adders as a flow-path to the program counter.

If multiple tagging is specified with this instruction, a successful reduction replaces the index registers with their OR'ed values minus the decrement. If a successful reduction is not possible, the index registers retain their original contents.

500.57

TRANSFER ON NO INDEX TNX -2000 I No Overlap (Figure 8)

The TNX instruction compares its decrement with the contents of the specified index register. If the number in the index register is greater than the decrement, the contents of the index register are reduced by the amount of the decrement and the computer proceeds to the next instruction in sequence. When the number in the index register is equal to or less than the decrement, no reduction is made and the computer transfers to location Y.

The sequence of operation for this instruction is similar to TIX except that the conditional transfer circuits are activated when there is a carry from XAD3.

500.54

TRANSFER ON INDEX HIGH TXH +3000 I No Overlap (Figure ~~18~~ 500.57)

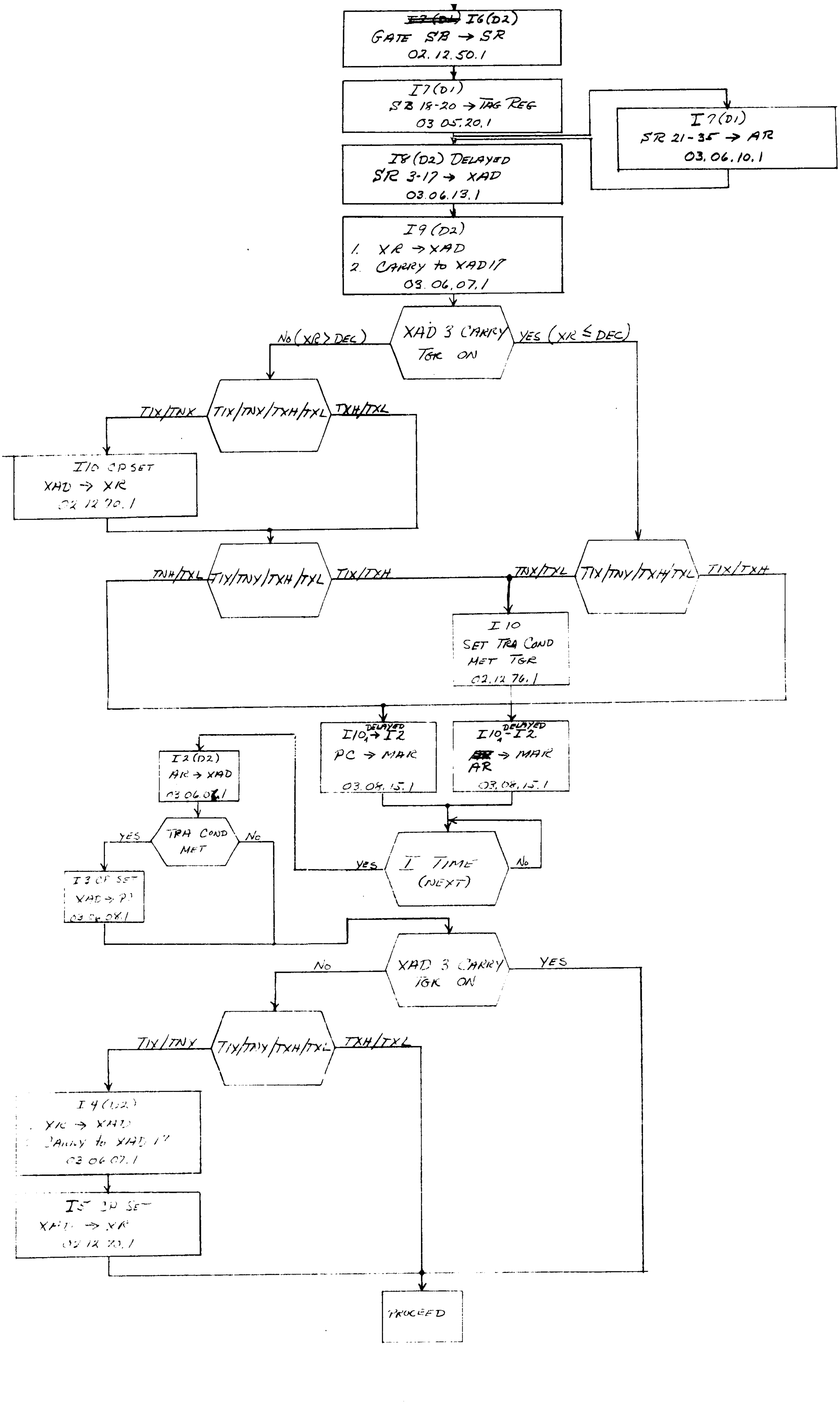
The TXH instruction compares its decrement with the contents of the specified index register. If the number in the specified index register is greater than the decrement, the computer transfers to location Y. If the number in the index register is less than or equal to the decrement, the computer takes the next instruction in sequence. Index register contents are not altered.

Execution of TXH is similar to TIX; at I8(D2) time delayed, ~~I9(D2) time~~ SR positions 3 - 17 are added to the 2's complement from the index register to determine if there will be a ripple carry from XAD3. This carry is the only indication needed; the index adders are not returned to the index register. A carry causes the program counter to be gated to MAR and the computer proceeds in sequence; no carry (index register high condition) causes the address register to be gated to MAR and the computer transfers to location Y by gating the address register through the index adders and into the program counter during the next I time.

TRANSFER ON INDEX LOW OR EQUAL TXL -0300 I No Overlap (Figure) pg 500.57

The TXL instruction compares its decrement with the contents of a specified index register. If the number in the index register is greater than the decrement, the computer takes the next instruction in sequence. If the number in the index register is equal to or less than the decrement, the computer transfers to location Y.

Execution of this instruction is similar to TNX except that the index adders are not returned to the index registers, An XAD3 carry (index register equal or low condition) indicates a successful transfer and causes the address register to be gated to MAR. No carry causes the program register to be gated to MAR and the computer proceeds to the next instruction in sequence.



~~I2 (D1)~~ I6 (D2)
GATE SB → SR
02.12.50.1
I7 (D1)
SB 18-20 → TAG REG
03 05.20.1
I7 (D1)
SR 21-35 → AR
03.06.10.1
I8 (D2) DELAYED
SR 3-17 → XAD
03.06.13.1
I9 (D2)
1. XR → XAD
2. CARRY to XAD 17
03.06.07.1
XAD 3 CARRY TGR ON
No (XR > DEC)
YES (XR ≤ DEC)
TIX/TNX/TXH/TXL
TIX/TNX
TXH/TXL
I10 CP SET
XAD → XR
02 12 70.1
TIX/TNX/TXH/TXL
TNH/TXL
TIX/TXH
TIX/TNX/TXH/TXL
TNX/TXL
TIX/TXH
I 10
SET TRA COND MET TGR
02.12 76.1
I10 DELAYED → I2
PC → MAR
03.08.15.1
I10 DELAYED - I2
~~SR~~ AR → MAR
03.08.15.1
I2 (D2)
AR → XAD
03 06.07.1
TRA COND MET
YES
No
I 3 CP SET
XAD → PC
03 06 08.1
I TIME (NEXT)
yes
No
XAD 3 CARRY TGR ON
No
YES
TIX/TNX/TXH/TXL
TIX/TNX
TXH/TXL
I 4 (D2)
1. XR → XAD
2. CARRY to XAD 17
03 06 07.1
I5 CP SET
XAD → XR
02 12 70.1
PROCEED

TRANSFER WITH INDEX INCREMENTED TXI +1000 I No Overlap (Figure pg 500.60)

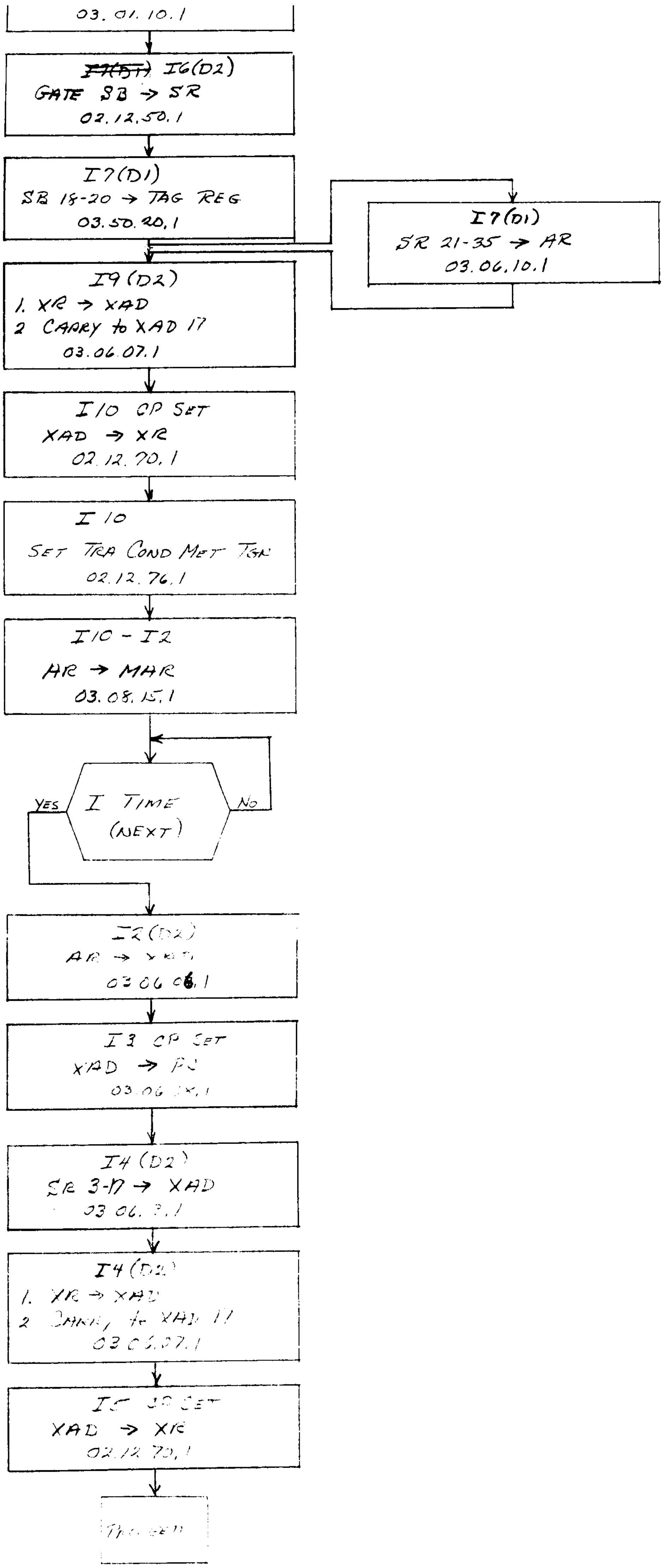
TXI is an unconditional transfer to the location specified in positions 21 - 35 of its address field. In addition, the decrement portion is added to the specified index register. This is a one (I) cycle instruction; therefore, overlapping is not permitted by an instruction in the next higher odd core storage location, Bit recognition in positions 1 or 2 of the storage bus (02.11.40.1) blocks sending positions 3 - 11 to the program register; instead, positions 1 and 2 are gated to PR positions 8 and 9. Decoding (02.12.76.1) recognizes this as a non-indexable instruction.

At I6(D2) and I7(D1) times respectively, ~~I7 time~~, the storage register and tag register are set from the storage bus; SR positions 21 - 35 are immediately sent to the address register to provide the transfer address when the AR is gated to MAR at I10 time. Because this is a non-indexable instruction, normal AR to XAD gating is blocked.

Index register contents always come out in complement form and if added to a number at this time would effectively accomplish subtraction. Because of this, it is necessary to cycl~~y~~e the index register through the index adders with a carry to XAD 17. This is accomplished at I9(D2) time and places the 2's complement of the origional value into the index register.

Because this is an unconditional type transfer instruction, the "transfer conditions met" trigger is turned ON at I10 time and the contents of the address register are gated to MAR. During the following I2(D2) time the address register is gated to the index adders and from there into the program counter.

During the next (I) cycle incrementing of the index register is accomplished by gating positions 3 - 17 (decrement portion) of the storage register to the index adders together with the index register and a carry to XAD 17. Setting the index adders back into the index register, completes the operation.



TXI FLOW CHART

Figure

TRANSFER AND SET INDEX TSX +0074 I No Overlap (Figure pg 500.63)

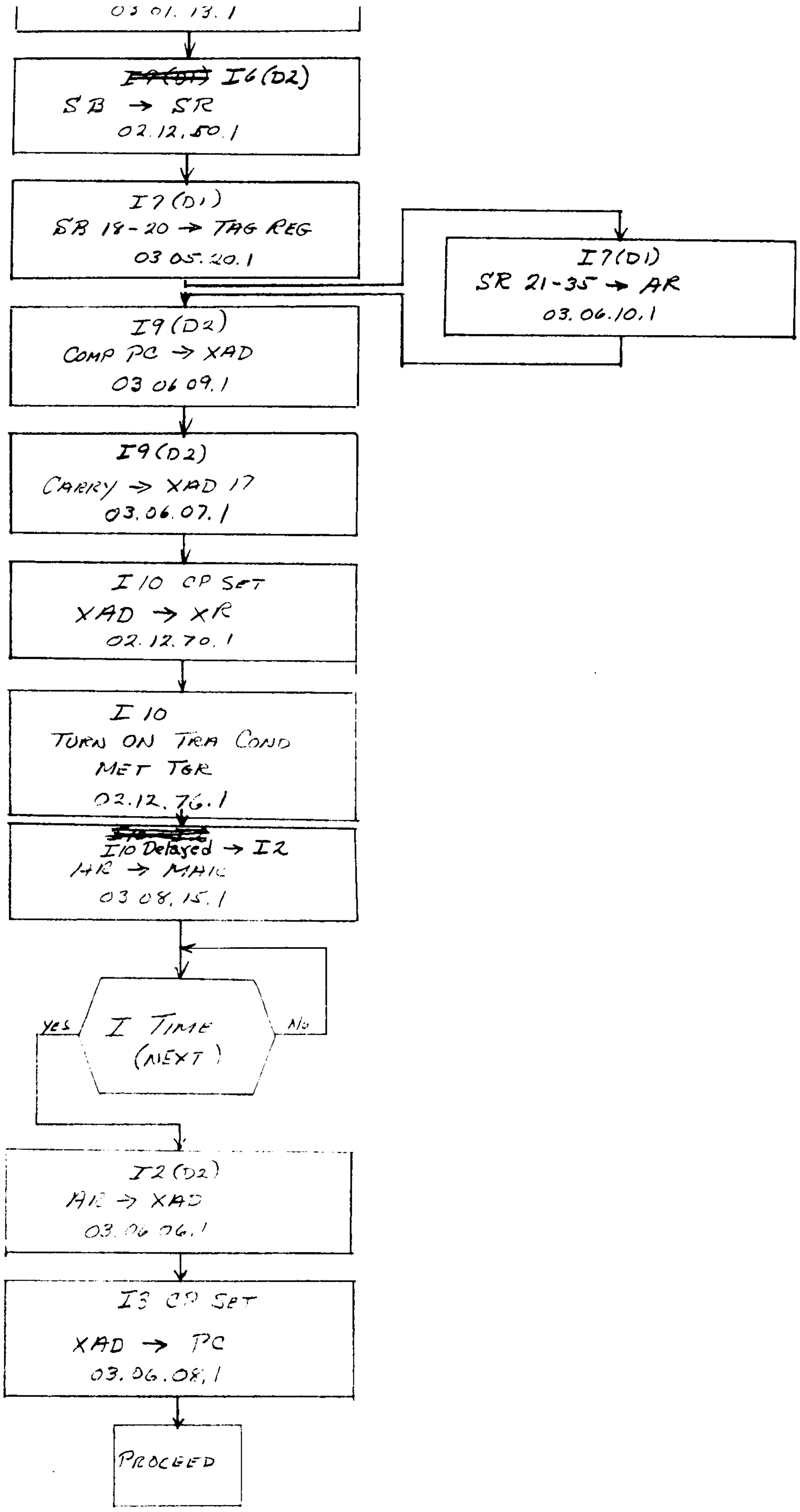
The TSX instruction places the 2's complement of the program counter (the location of the TSX instruction) into the specified index register and transfers to the location specified in positions 21 - 35 of the TSX instruction. This is a one (I) cycle instruction; therefore, overlapping is not possible by an instruction in the next higher odd core storage location. Decoding of PR 6 and 7 (02.12.76.1) defines this as a non-indexable instruction.

At I6(D2) and I7(D1) times respectively, ~~I7 time~~ the storage register and tag register are set from the storage bus, and SR positions 21 - 35 immediately sent to the address register. This is the transfer address which is used as the core storage reference for the next cycle.

To store the present value of the program counter, the normal stepping circuitry which occurs during I7(D2) time must be blocked. The program counter is gated to the index adders in the usual manner but the index adders are blocked from being routed back again to the program counter.

The program counter can be gated to the index adders in either true or complement form. The 2's complement is made available to the index register by gating the complement of the program counter to the index adders with a carry to XAD 17. Normal gating of the address register to the index adde at I9(D2) time is blocked by TSX (03.06.07.1); normal gating of the address register to the index adders is blocked because this is a non-indexable instruction (03.06.06.1).

At I10 time the "transfer conditions met" trigger is unconditionally set ON and the address register is gated to MAR. During the early portion of the next (I) cycle, the address register value is gated to the index adders and set into the program counter to complete the operation.



TSX FLOW CHART

508

~~5.3.11~~ AND and OR Instructions

The AND and OR instructions are used to produce logical combinations of bits. These are useful for masking or matching words.

OR to Storage ORS -0602 (I, E) Figure ~~5.3.67~~ 508-1
SLA

This instruction stores the logical OR of the word stored at location X and the AC (P, 1-35) in location X. The logical OR is obtained by matching the two words and storing ones in all positions which have ones in either word. For this instruction, the OR is developed in the memory data register during the E cycle. The words from core storage and the SB both go the memory data register and the result in the memory data register is put back into core storage.

OR to Accumulator ORA -0501 (I, E) Figure ~~5.3.68~~ 508-2
TLA or DLA

This instruction places the logical OR of the word stored at location X and AC(P-35) in the accumulator. The OR is produced by matching the two words and placing ones in all positions of the AC which have ones in either word. The OR is developed in the SR by gating both the storage and AC words into the SR at the same time. S and Q remain unchanged.

AND to Accumulator ANA -0320 (I, E, L) Figure ~~5.3.69~~ 508-3
TLA or DLA

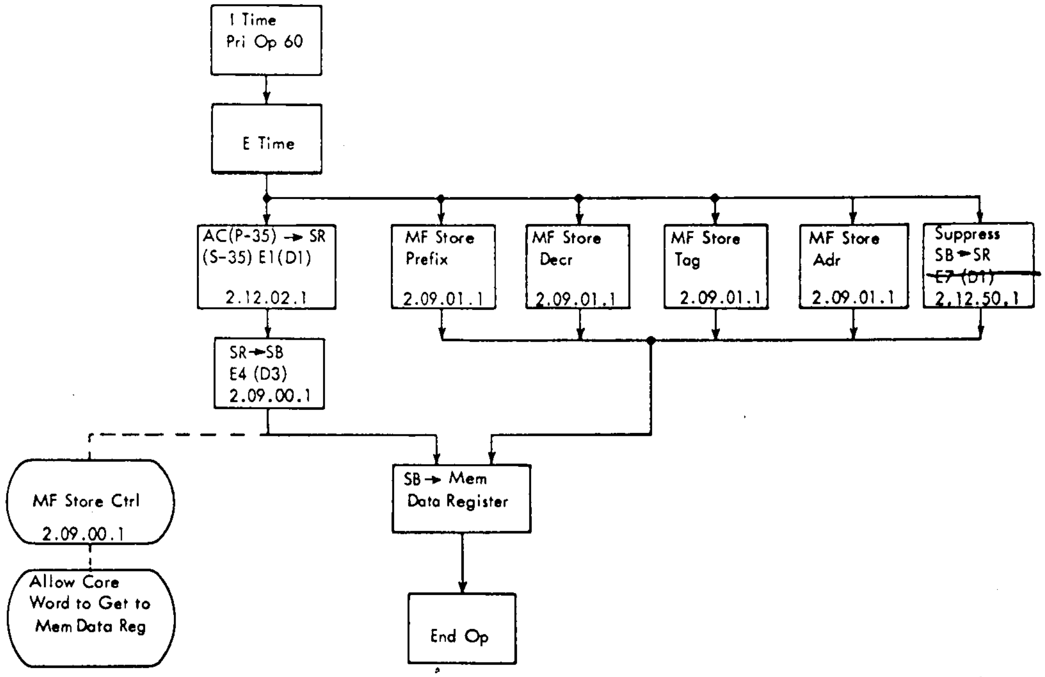
This instruction places the logical AND of the word stored at location X and the AC (P, 1-35) in the accumulator (P, 1-35). The logical AND is obtained by matching the two words and placing ones in all positions of the accumulator which have corresponding ones in both the AC and storage. This instruction is executed by OR'ing the complement of both words and then complementing this sum.

AND to Storage ANS +0320 (I, E, L, E) Figure ~~5.3.69~~ 508-3
TLA or DLA

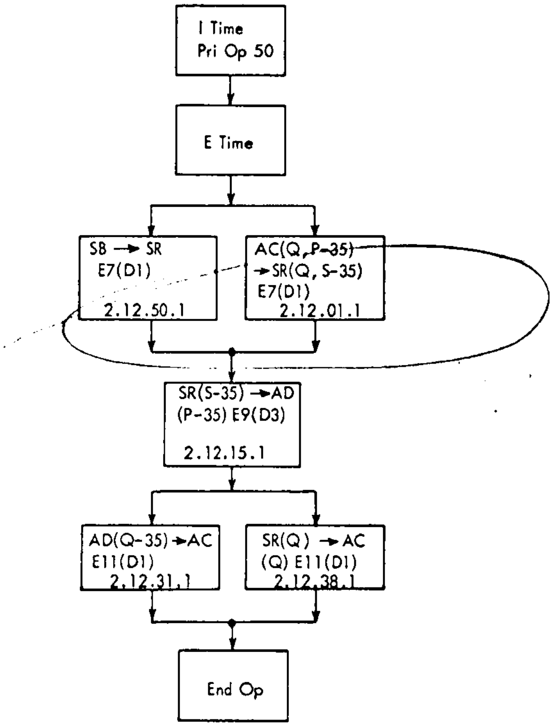
This instruction stores the logical AND of the word stored at location X and the contents of the AC(P, 1-35) in location X. The logical AND is obtained by matching the two words and placing ones in all positions of storage location X which have corresponding ones in both the AC and storage. The contents of the AC(S, Q, P, 1-35) are unchanged. The AND is accomplished the same for this instruction as for ANA. An additional E cycle is required to put the AND in storage. The complement of the original AC (P, 1-35) is returned to the AC and re-complemented to restore the contents of AC(P, 1-35). During the execution, the original AC(Q) is saved in SR(Q) and then returned to AC(Q).

Exclusive OR to Accumulator ERA +0322 (I, E, L) Figure ~~5.3.70~~ 508-4
TLA or DLA

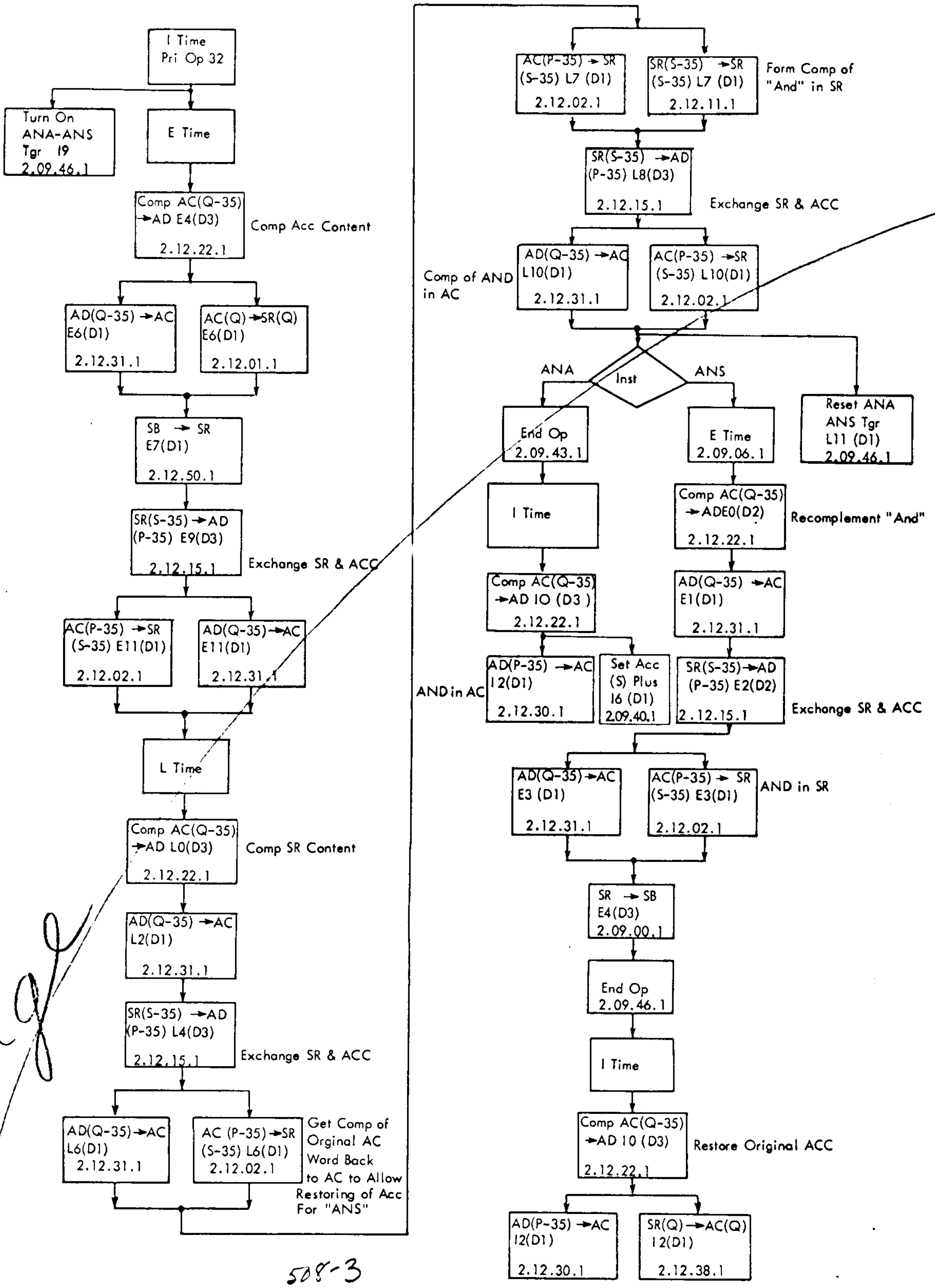
This instruction matches AC(P, 1-35) with the logical word stored at location X. Zeros replace all positions of the AC which match the corresponding positions of the logical word. Accumulator positions (S) and (Q) are cleared.



508-1
FIGURE ~~5.2-6~~7. ORS - 0602



508-2
FIGURE ~~5.2-6~~8. ORA - 0501



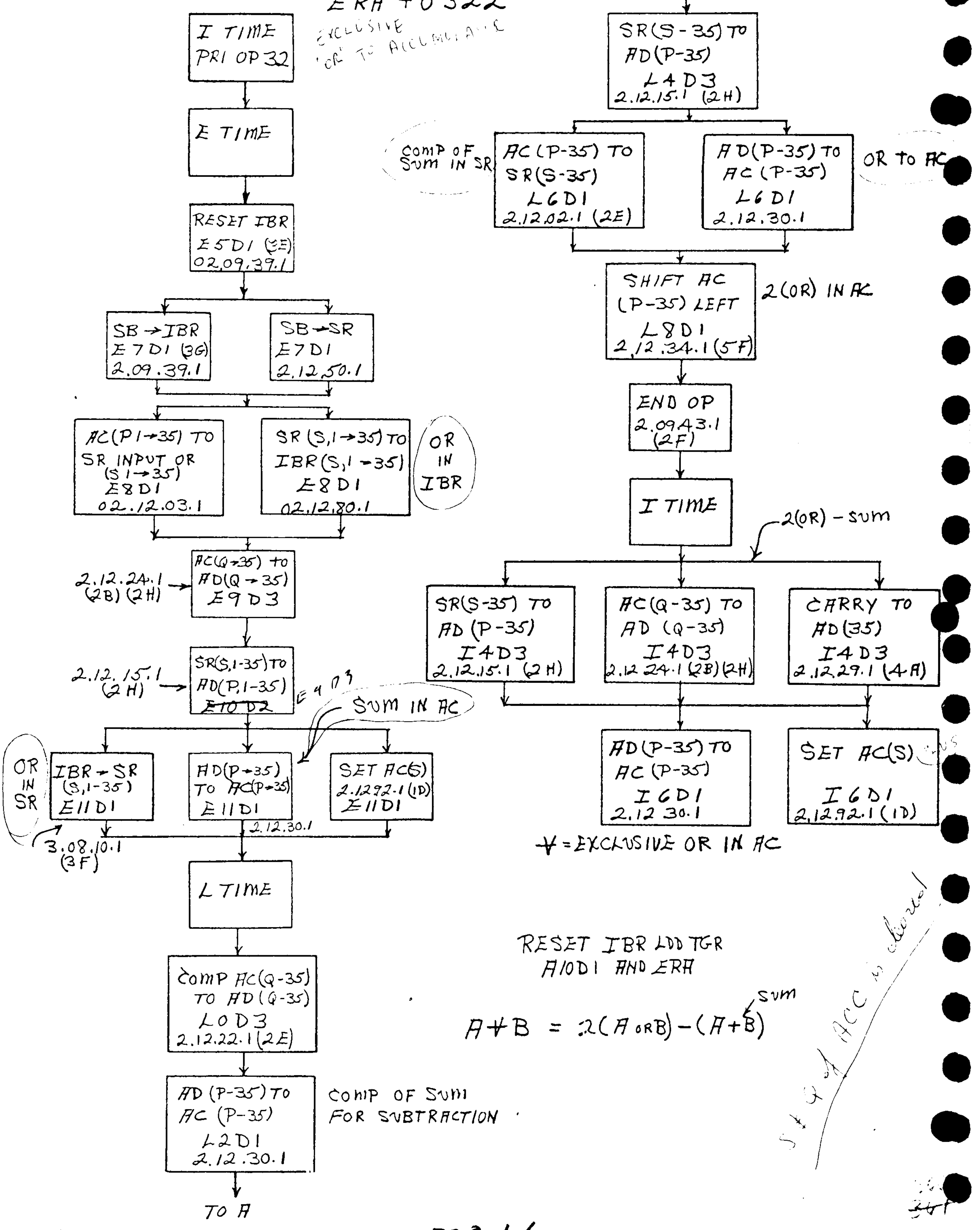
change

508-3

FIGURE ~~5.3-59.~~ ANA - 0320; ANS +0320

ERA +0322

EXCLUSIVE 'OR' TO ACCUMULATE



$$A \veebar B = 2(A \text{ or } B) - (A + \overset{\text{sum}}{B})$$

500.66

Logically speaking, the EXCLUSIVE OR is the result of OR'ing bits from either one word or the other, but not both. The computer, which can only AND and OR, makes use of the following logical equation to develop the EXCLUSIVE OR:

EXCLUSIVE OR = 2 (A or B) - (A + B)

The derivation of the above equation can be shown with the following adder table:

Factor A	0	0	0	1	1
Factor B	0	0	1	0	1
A + B (carry not blocked)	0	1	0	0	0
Carry to be blocked	0	0	0	1	0
A + B (carry blocked)	0	0	1	1	0
EXCLUSIVE OR (A + B carry blocked)	0	0	1	1	0
A or B	0	0	1	1	1
2 (A + B)	1	0	0	0	0
2 (A or B)	0	1	1	1	0

From the table it can be seen that the EXCLUSIVE OR is equal to the sum output of the adders with all carries blocked. The carries cannot be blocked, but the EXCLUSIVE OR can be obtained by subtracting an amount equal to the blocked carries from the sum of two words:

1. EXCLUSIVE OR = (A + B) - blocked carries
   The blocked carries can be simulated by subtracting twice the OR from twice the sum of two words:
2. Blocked carries = 2 (A + B) - 2 (A or B) = 10000 - 01110 = 00010
   Substituting the equation 2 in equation 1:
3. EXCLUSIVE OR = (A + B) -[2 (A + B) - 2 (A or B] = 01000 - 00010 = 00110
   And simplifying equation 3:
4. EXCLUSIVE OR = 2 (A or B) - (A + B) = 01110 - 01000 = 00110

500.67

509 ~~5.3.12~~ Convert Instructions

The three convert instructions can materially reduce the time required for many "housekeeping" and table-look-up routines. They can be used for number conversions, for preparing print fields, and even for adding numbers in systems other than binary.

The convert instructions, like variable-length instructions, include a count field as well as the operation code, address and tag bit. The convert instructions cause a series of references to be made (usually six) and the address specifies the starting location of the first storage table. The register (accumulator or MQ) from which the reference is controlled is considered to be made up of six 6-bit groups. The first of these groups is added to the instruction address to give the location of the first storage reference. The word stored at this location must contain, in addition to its conversion information, the starting location of the next storage table. The convert-by-replacement instructions shift the controlling register six places, clearing the six places on the opposite end of the register. Positions (S-5)* of the table word are entered into the six cleared positions, and the next group of six bits is in position to add to the starting location of the next table. The process continues until up to six references have been made. The controlling register is gradually replaced by six-bit entries from the storage tables. When the number of references specified by the count has been made, the conversion is complete.

The tag of the instruction has a unique function for the convert instructions. Positions (18) and (19) are not used, but a bit in (20) will cause the storage table starting location contained in the last reference word to be stored in index register A.

Convert by Replacement from Accumulator   CVR +0114 (Min I, L) Figure ~~5.3.71~~ 509-1
(Max I, L, 6E)

This instruction treats the contents of the AC(P, 1-35) as six 6-bit representations. The instruction replaces a number of these representations, equal to the count of the instruction, with the contents of positions (S-5) of a like number of words from storage. These words are found by adding AC(30-35) (the first six-bit group) to SR(21-35) (initially the instruction address) and directing storage to this modified address. The word thus found is brought to the SR; the AC is shifted right six places; SR(S-5) replaces AC(P, 1-5). SR(21-35) adds to the next six-bit group in AC(30-35) to locate the next word in storage. The process is then repeated. After the required number of replacements, the address portion of the last storage word can be stored in index register A by including a "tag" bit in position (20) of the instruction.

The following illustrates the use and operation of CVR:

Direct Addition of BCD Numbers:

A	+	B	=	C
134589	+	691593	=	826182

Table required in storage for this example:

Storage Location	Contents	
	(S-5)	(21-35)
1000	0	1000
1001	1	1000
1002	2	1000
1003.	3	1000
1004	4	1000
1005	5	1000
1006	6	1000
1007	7	1000
1008	8	1000
1009	9	1000

Storage Location	Contents	
	(S-5)	(21-35)
1010	0	1001
1011	1	1001
1012	2	1001
1013	3	1001
1014	4	1001
1015	5	1001
1016	6	1001
1017	7	1001
1018	8	1001
1019	9	1001

Instructions required for operation:

```
CAL  A           (First BCD word)
ADD  B           (Second BCD word)
CVR  6, 0, 1000  (Convert sum to BCD word)
```

* This text section uses (S-5) to represent (S, 1-5).

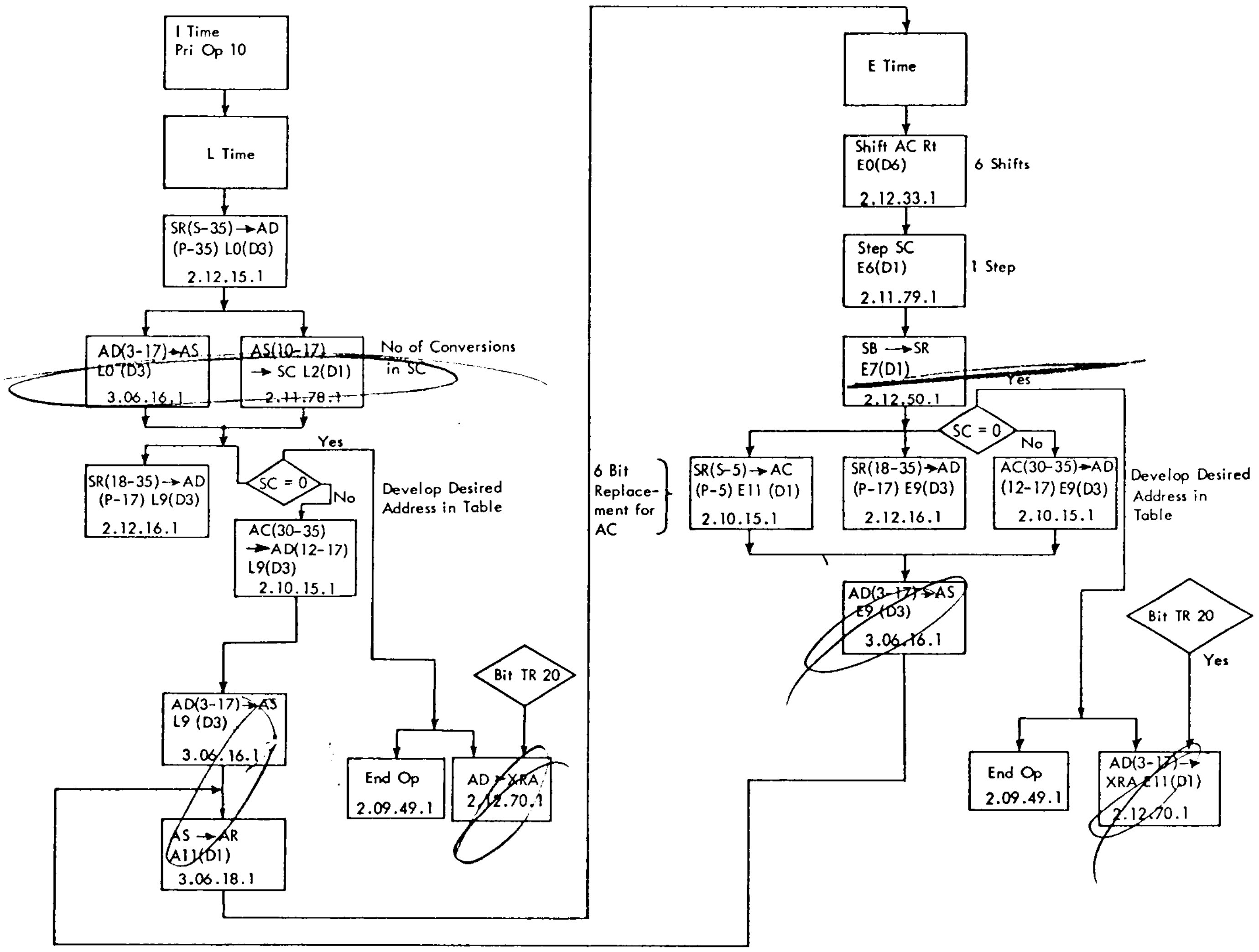


FIGURE ~~5.3-7~~ 509-1. CVR +0114

Development of Program

CAL A(Clear AC, add logical word A)→
Add B
(Unconverted binary sum)
CVR 1000, 0, 6

	CONTENTS OF ACCUMULATOR						
	P1-5	6-11	12-17	-23	-29	-35	
CAL A	1	3	4	5	8	9	1st
Add B	+6	9	1	5	9	3	Char
(Unconverted binary sum)	=7	12	5	10	17	12	Conv
							ARS 6
	2	7	12	5	10	17	Count = 5
	8	2	7	12	5	10	= 4
	1	8	2	7	12	5	= 3
	6	1	8	2	7	12	= 2
	2	6	1	8	2	7	= 1
	8	2	6	1	8	2	= 0

Series of Steps within CVR

C(AC) 30-35		Start Loc		Table Refer	Table C(-) 1-5	Next Start Loc C(-) 21-35
*12	+	1000	-	1012	2	1001
17	+	1001	-	1018	8	1001
1'0	+	1001	-	1011	1	1001
5	+	1001	-	1006	6	1000
12	+	1000	-	1012	2	1001
7	+	1001	-	1008	8	1000

Count = zero: if Tag = 1, 1000 XRA
SLW C (Store converted BCD sum as logical word)

The execution of CVR requires one L cycle to set the count in the shift counter and to calculate the initial table location. The instruction is completed in as many E cycles as specified by the count.

Convert by Replacement from MQ CRQ -0154 (Min I, L) Figure ~~5-3-72~~ 509-2
(Max I, L, 6E)

This instruction operates on the MQ, considered to be composed of six 6-bit representations. The instruction replaces a number of these representations equal to the count of the instruction, with the contents of positions (S-5) of a like number of words from storage. The location of the first of these storage words is found by adding the contents of MQ(S-5) to SR(21-35) (intially the instruction address portion). The word stored at this modified location is brought to the SR, and the MQ is shifted left six places. Positions (S-5) of the stored word are placed in MQ(30-35), and the location of the next storage word is computed by again adding MQ(S-5) to SR(21-35) which is now the address portion of the previous storage word. The process continues until the required number of replacements have been made. At this time the presence of a tag bit in position 20 of the instruction will cause the address portion of the final storage word to be stored in index register A.

The following illustrates the use and operation of CRQ:

Prepare a BCD number for printing be replacing leading zeros with blanks:

Convert BCD number 007109 to BL BL 7109
Instructions required for this operation:

LDQ	A	(BCD number in storage)
CRQ 6, 0, 2000		(Convert from MQ)
STQ	B	(Store converted number)

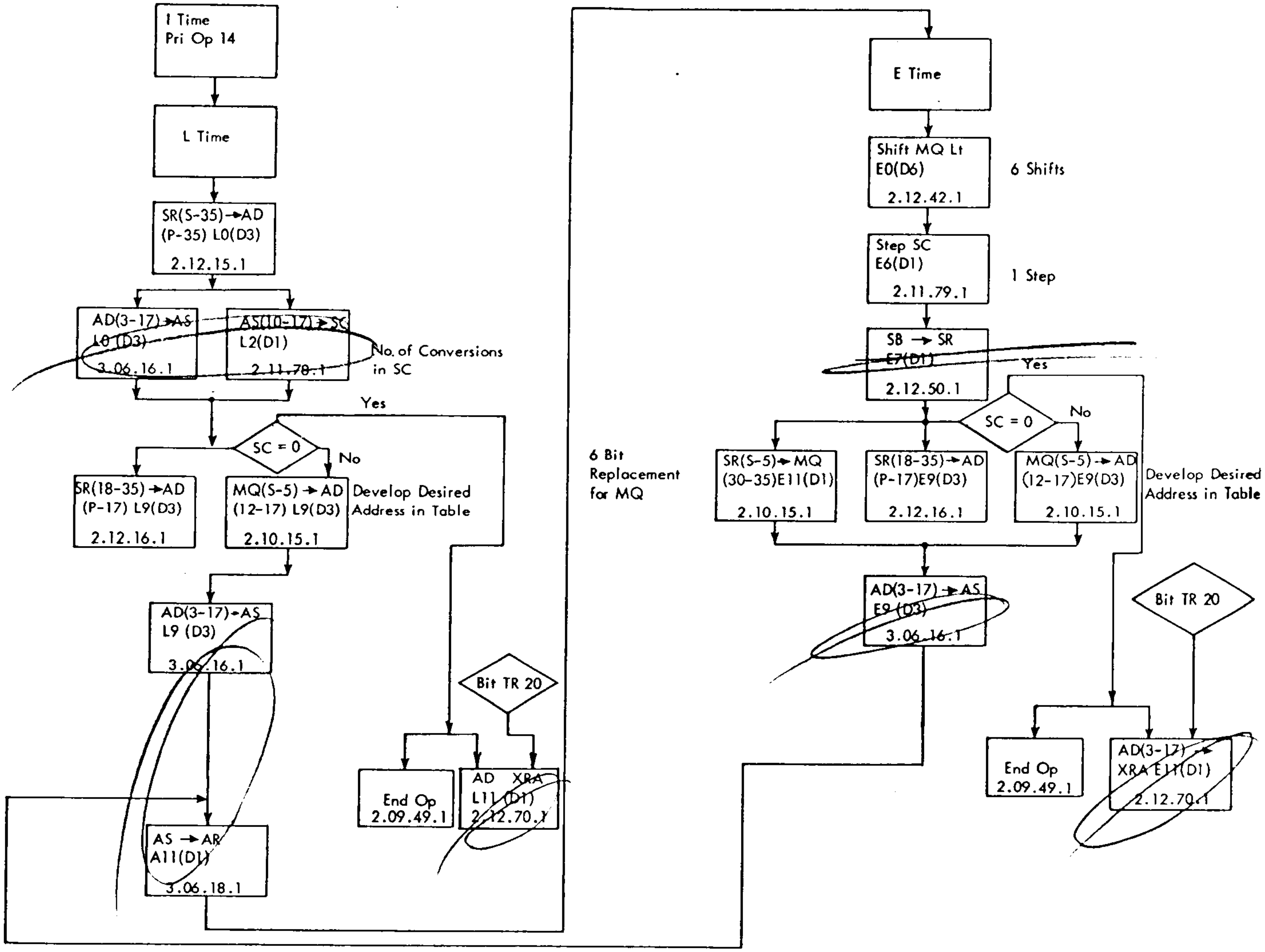


FIGURE ~~5.3-7~~2. CRQ - 0154
509-2

Storage table required for CRQ (in decimal):

Storage Location	Contents	
	(S-5)	(21-35)
2000	BL	2000
2001	1	2010
2002	2	2010
2003	3	2010
2004	4	2010
2005	5	2010
2006	6	2010
2007	7	2010
2008	8	2010
2009	9	2010

Storage Location	Contents	
	(S-5)	(21-35)
2010	0	2010
2011	1	2010
2012	2	2010
2013	3	2010
2014	4	2010
2015	5	2010
2016	6	2010
2017	7	2010
2018	8	2010
2019	9	2010

Development of Program

LDQ A

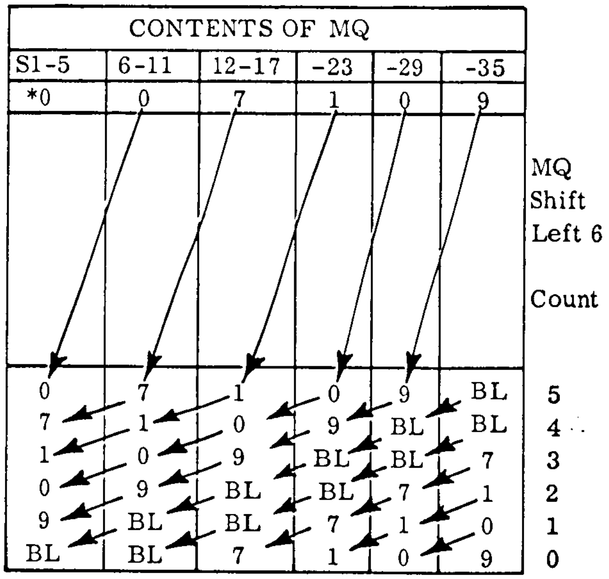
CRQ 2000, 0, 6

1st Char Conv

Series of Steps within CRQ

C(MQ)s 1-5		Start Loc	Table Refer	Table C(-)s 1-5	Next Starting Loc C(-) 21-35
*0	+	2000 =	2000	BL	2000
0	+	2000 =	2000	BL	2000
7	+	2000 =	2007	7	2010
1	+	2010 =	2011	1	2010
0	+	2010 =	2010	0	2010
9	+	2010 =	2019	9	2010

Count = 0; if Tag = 1, 2010 →XRA



CONTENTS OF MQ

S1-5	6-11	12-17	-23	-29	-35	MQ Shift Left 6 Count
*0	0	7	1	0	9	
0	7	1	0	9	BL	5
7	1	0	9	BL	BL	4
1	0	9	BL	BL	7	3
0	9	BL	BL	7	1	2
9	BL	BL	7	1	0	1
BL	BL	7	1	0	9	0

The execution of CRQ is accomplished in one L cycle and a number of E cycles equal to the count.

Convert by Addition from MQ CAQ -0114(Min I, L) (Max I, L, 6E) Figure ~~5-3-73~~ 509-3

This instruction treats the MQ as six 6-bit representations, as does CRQ. This instruction does not replace any of these representations, but uses them to locate a number of storage words equal to the count of the instruction. The storage words are located at the address developed by adding MQ(S-5) to SR(21-35) (initially the instruction address) and are brought to the SR. The storage words are then added to the contents of the accumulator. The MQ is not replaced, but is merely rotated left six places to allow the next six-bit representation to be added to the address portion of the previous storage word. The process continues until a number of additions (to the AC) equal to the count have been made. The operation is then complete. The address portion of the last storage word used can be stored in index registers for future reference by adding a tag bit in position 20 of the instruction.

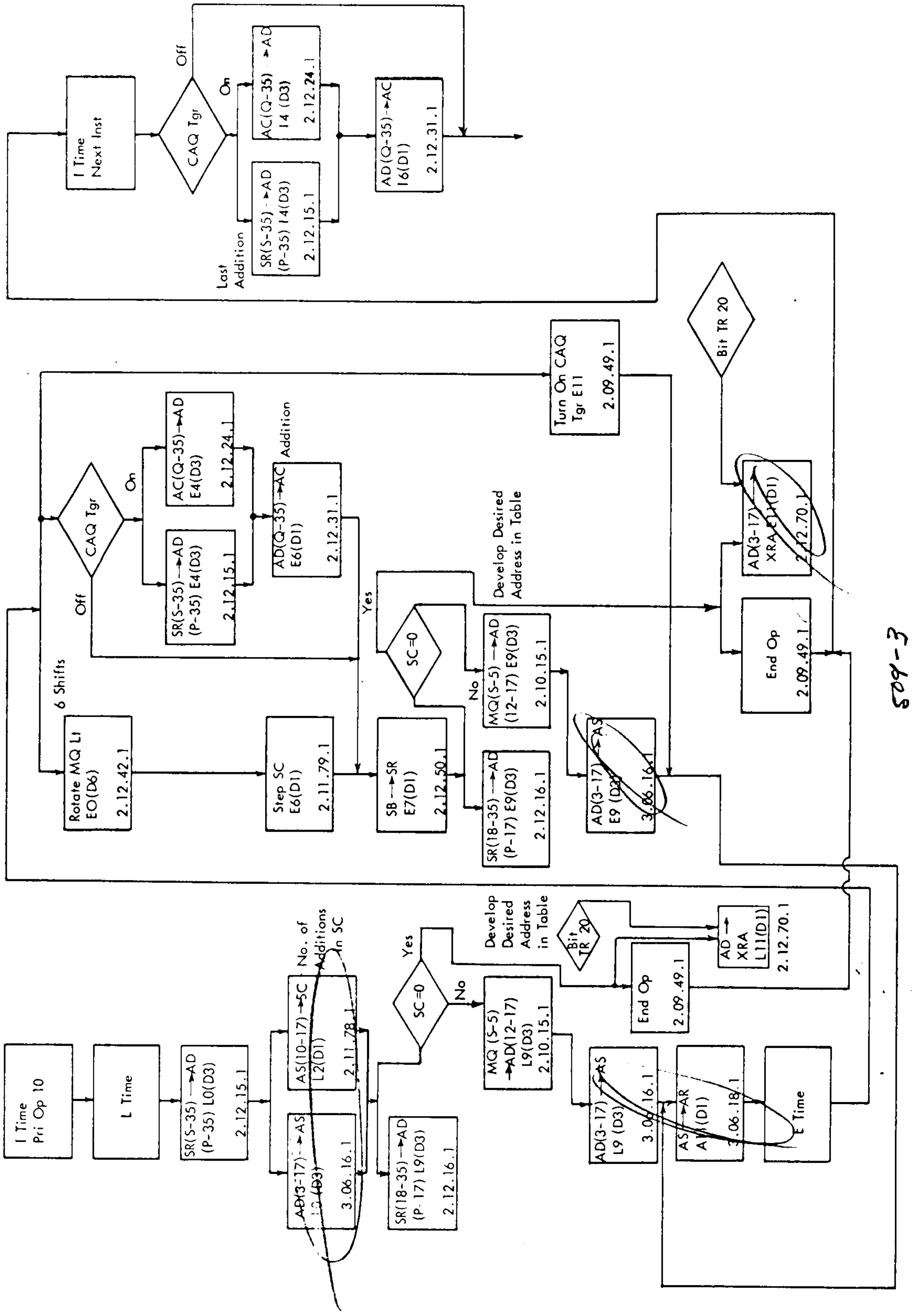


FIGURE 509-3. CAQ-0114

To illustrate the operation and use of CAQ, a program which will convert BCD to binary follows.

Convert BCD Word to Binary

709542 (BCD) converted to 2,551,646 (Octal)

Instructions required for this operation:

LDQ	A	(BCD word)
CLM		(Clear AC)
CAQ 6, 0, 3000		(Convert to binary)
ARS 16_{10}		(Position result in AC)
SLW	B	(Store result)

Storage table required for CAQ:

Storage Location	Contents	
	(S-19)	(21-35)
Decimal	Octal	Decimal
3000	0	3100
3001	303,240	3100
3002	606,500	3100
3003	1,111,740	3100
3004	1,415,200	3100
3005	1,720,440	3100
3006	2,223,700	3100
3007	2,527,140	3100
3008	3,032,400	3100
3009	3,335,640	3100
3100	0	3200
3101	23,420	3200
3102	47,040	3200
3103	72,460	3200
3104	116,100	3200
3105	141,520	3200
3106	165,140	3200
3107	210,560	3200
3108	234,200	3200
3109	257,620	3200
3200	0	3300
3201	1,750	3300
3202	3,720	3300
3203	5,670	3300
3204	7,640	3300
3205	11,610	3300
3206	13,560	3300
3207	15,530	3300
3208	17,500	3300
3209	21,450	3300

Storage Location	Contents	
	(S-19)	(21-35)
Decimal	Octal	Decimal
3300	0	3400
3301	144	3400
3302	310	3400
3303	454	3400
3304	620	3400
3305	764	3400
3306	1,130	3400
3307	1,274	3400
3308	1,440	3400
3309	1,604	3400
3400	0	3500
3401	12	3500
3402	24	3500
3403	36	3500
3404	50	3500
3405	62	3500
3406	74	3500
3407	106	3500
3408	120	3500
3409	132	3500
3500	0	
3501	1	
3502	2	
3503	3	
3504	4	
3505	5	
3506	6	
3507	7	
3508	10	
3509	11	

Development of Program

Binary Equivalent

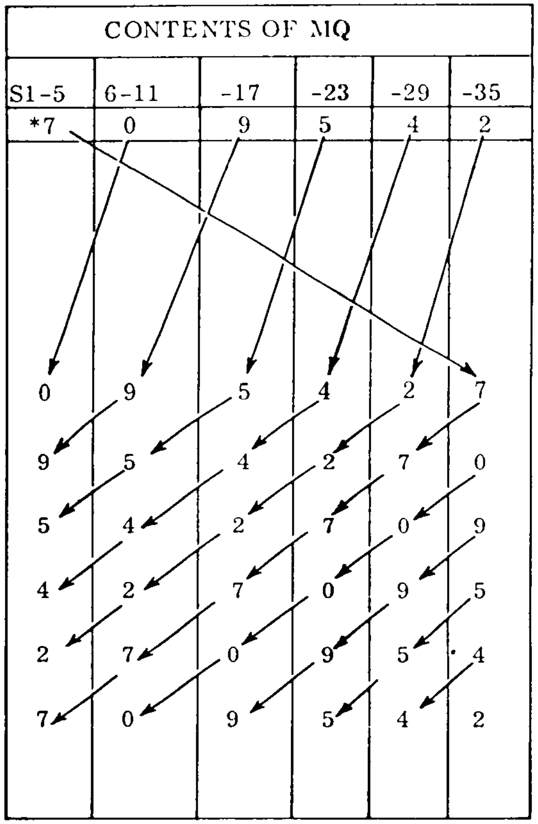
BCD Word

	CONTENTS OF ACCUMULATOR	
	P, 1-19	21-35
LDQ BCD WORD	XX	XX
CLM	00	00
CAQ 3000, 0, 6		RQL 6

Series of Steps within CAQ

C(MQ) S, 1-5		Start Loc		Table Refer	Table C(-)S, 1-19	Next Start Loc C(-) 21-35)		P, 1-19	21-35	Count
7	+	3000	=	3007	2,527,140	3100	+	2,527,140	3100	5
0	+	3100	=	3100	0	3200	+	0	3200	4
							=	2,527,140	6300	
9	+	3200	=	3209	21,450	3300	+	21,450	3300	3
							=	2,550,610	9600	
5	+	3300	=	3305	764	3400	+	764	3400	2
							=	2,551,574	13000	
4	+	3400	=	3404	50	3500	+	50	3500	1
							=	2,551,644	16500	
2	+	3500	=	3502	2	-	+	2	-	0
							=	2,551,646	16500	

ARS 16 (Binary Equivalent) ⟶ 2,551,646



CONTENTS OF MQ					
S1-5	6-11	-17	-23	-29	-35
*7	0	9	5	4	2
0	9	5	4	2	7
9	5	4	2	7	0
5	4	2	7	0	9
4	2	7	0	9	5
2	7	0	9	5	4
7	0	9	5	4	2

SLW_

7094 TRAPPING

6·1

~~5-3~~.06 Trap Mode Instructions

The 7090 can be operated in either of two modes, normal or trapping. Entrance to trapping mode is gained by executing the ETM instruction. Exit is accomplished by using the LTM instruction or the clear or reset keys on the console. Trapping mode affects the operation of transfer instructions except TTR. In trapping mode, the location of each transfer instruction is stored in the address portion of location 0000. Successful transfers are not executed; instead, an instruction transfer, or trap, is taken to location 0001. One instruction, trap transfer, is immune to trapping mode. The locating of successful transfers and the trap to a common check point make trapping mode useful for debugging program flow. Unsuccessful transfers have their location stored in the address portion of 0000, but do not trap to 0001.

Enter Trapping Mode — ETM +0760...0007 (I, L)

This instruction places the computer in trapping mode by turning on the trap mode trigger on Systems 2.10.53.1. The computer remains in trapping mode until a LTM instruction is executed or the clear or reset buttons are depressed. ETM is a primary operation 76 instruction and requires an I and an L cycle.

TRA in Trapping Mode — TRA +0020(I, E)

The effect of trapping mode on a transfer is shown in Figure ~~5.3-38.~~ 601-1

Leave Trapping Mode — LTM -0760...0007 (I, L)

This instruction returns the computer to normal mode by turning off the trap mode trigger on Systems 2.10.53.1. This is a primary operation 76 instruction, and an I and an L cycle are required.

Trap Transfer — TTR +0021 (I)

This is the only instruction which provides an instruction transfer to location X regardless of the operating mode of the computer. The TTR instruction nullifies the transfer blocking circuits of trapping mode (Systems 2.10.53.1) and operates like TRA.

Store Location and Trap — STR -1000(I, E) Fig~~ure 5.3-34~~ 601-2

This instruction stores its location plus one in the address portion of storage location 0000. It then traps or transfers the computer to loaction 0002 where the instruction sequence is resumed. STR does not place the computer in the trapping mode.

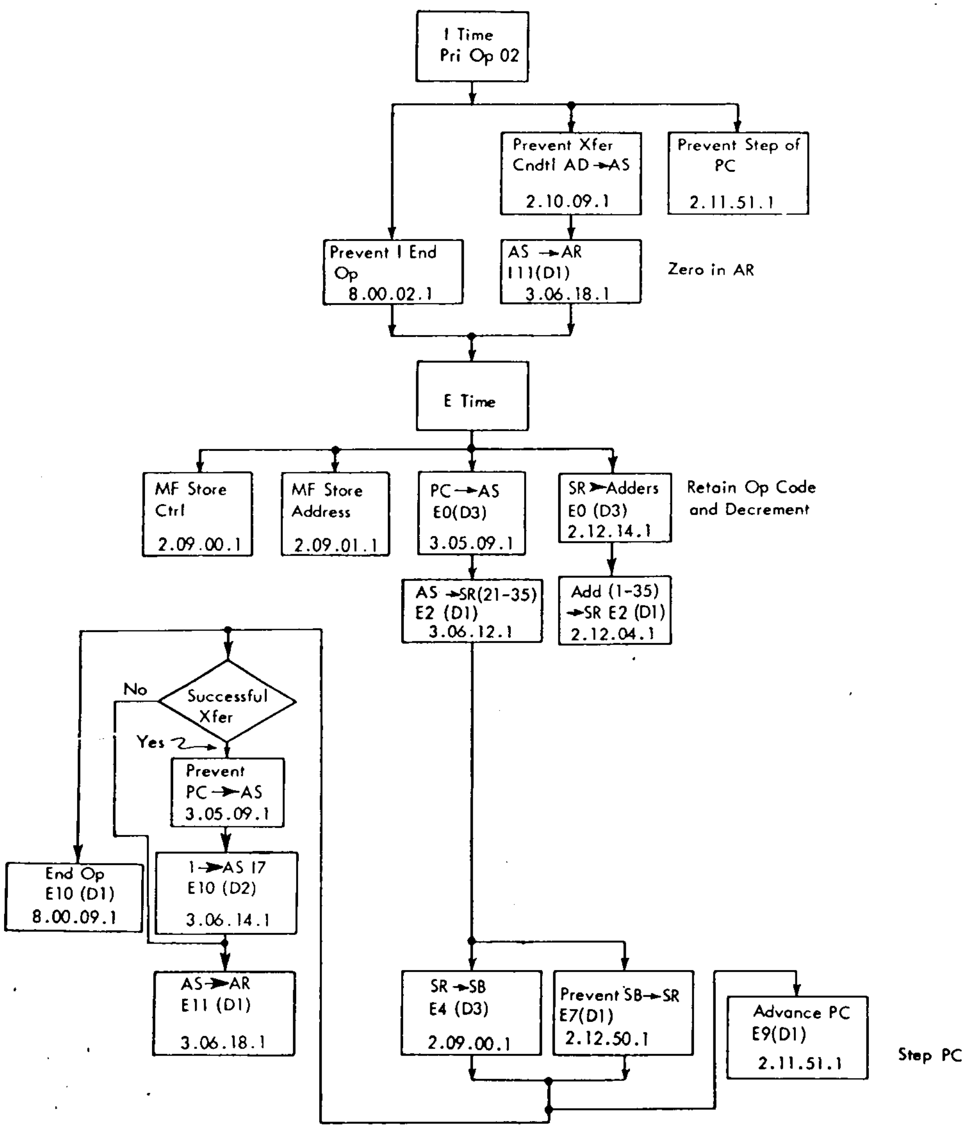


FIGURE ~~5.3-33.~~ TRA +0020 TRAP MODE

601-1

Fig 601-1

p 601.2

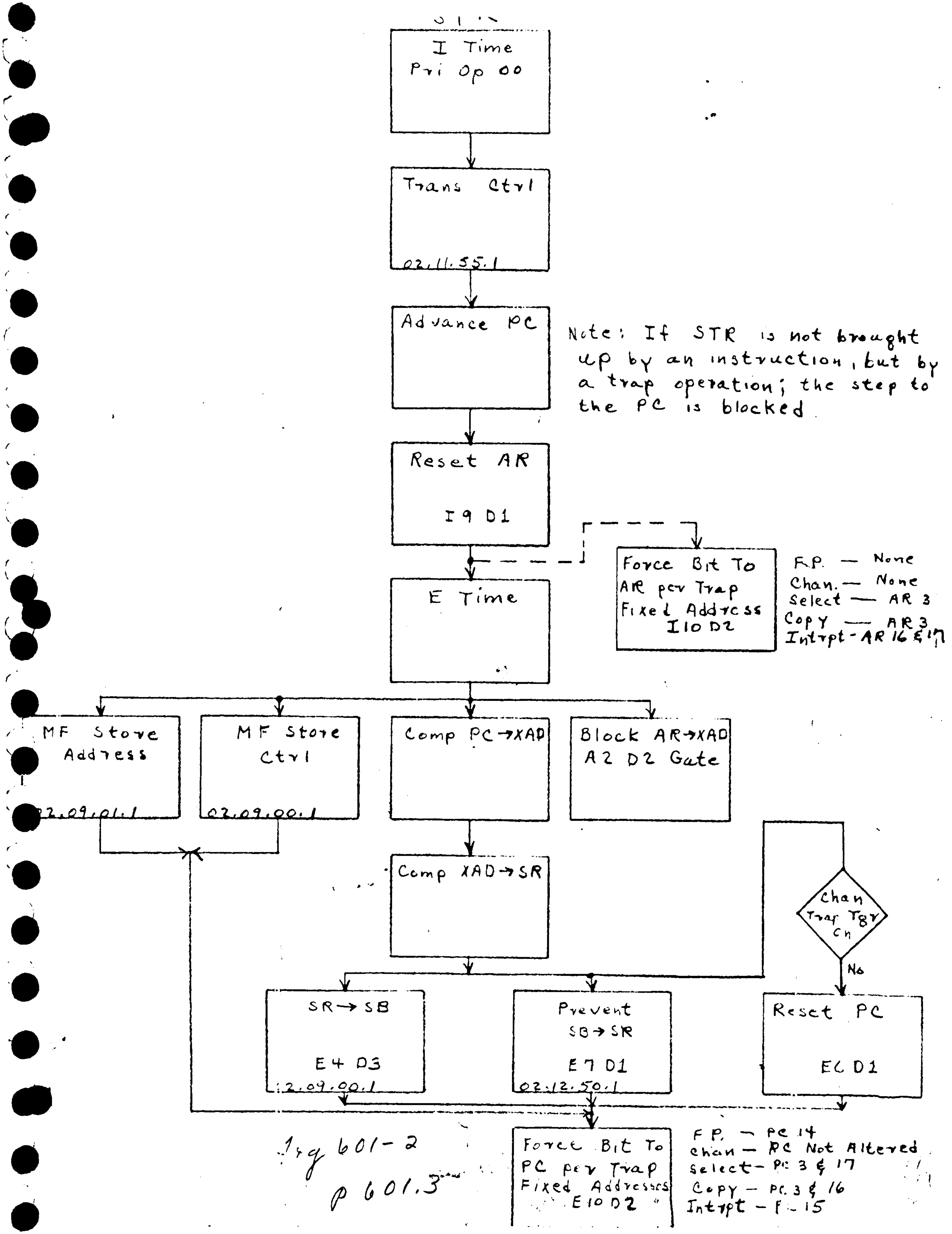


Fig 601-2

p 601.3

INSTRUCTION OVERLAP

The instruction overlap feature of the 7094 significantly reduces the number of machine cycles required for computer operation. Instruction overlap reduces the number of cycles for certain pairs of instructions from four to three. Certain other pairs of instructions may be performed in full overlap, i. e., no cycle time is required for the second instruction and the two instructions are performed in the time normally required for the first one. Proper programming techniques are necessary to realize the full benefits of the instruction overlap feature.

Instruction overlap takes advantage of the fact that core storage (memory) always offers two full words to the computer -- those words in the even and the next highest odd address location.It is the nature of memory that the full 72 bits of two locations, even and odd, must be strobed on every request to a core storage address. After strobing, the 72 bits are sent to the memory data register (MDR).

Either or both of the words in the MDR may be gated to the storage bus, as determined by the contents of the MAR (memory address register). bit position 17. If MAR 17 contains a 1, the reference must have been to an odd address (MAR 17 is the 2^0 position), and only the word in the odd address location will be gated out by the memory internal data out gate (DOG). When MAR 17 contains a 0, the reference must have been to an even address. The internal DOG will gate out the first word (even address) and a split DOG control line, originating in the computer, gates out the second word. This sequential gating out of two words occurs whenever the reference is to an even address and the computer isnot in a channel operation. When two words are gated to the computer, the first word comes off the storage bus into the storage register. The second instruction is gated into the instruction backup register (XX (IBR) for partial decoding. Occasionally, two words will come into the computer and that in the IBR will be ignored -- to be destoyed the next time the IBR is reset. This occurs when the word in the IBR, in SR does nct meet thecriteria established for instruction overlap.

Three kinds of overlap are possible:

1. Data lookahead (DLA) -- in which the need for an operand from core storage is anticipated.

2. Store lookahead (SLA) -- in which the need to referx to memory for a store operation is anticipated.

3. Transfer lookahead (TLA) -- in which a transfer function is anticipated.

The sequence of an instruction overlap begins with with an instruction xxxxlapx cycle reference to an even address in core storage. Notice from Figure 700 that bits 0 - 35 will be gated out of memory by the DOG. When the split DOG comes up in the computer the data out gate for bits 36 - 71 will be enabled and the gate for bits 0 - 35 disabled. The even-address word is gated into the computer at I6 time, the odd-address word at I9. Figure 701 shows the timing for the split DOG control line. Notice the split DOG is controlled by B time and the inputs at OR circuit 3G. These functions relate to data channel operation so it is impossible to have the split DOG control line up when the word from memory is destined for the data channel.

With the word on the SB it is still necessary to gate from the SB to the

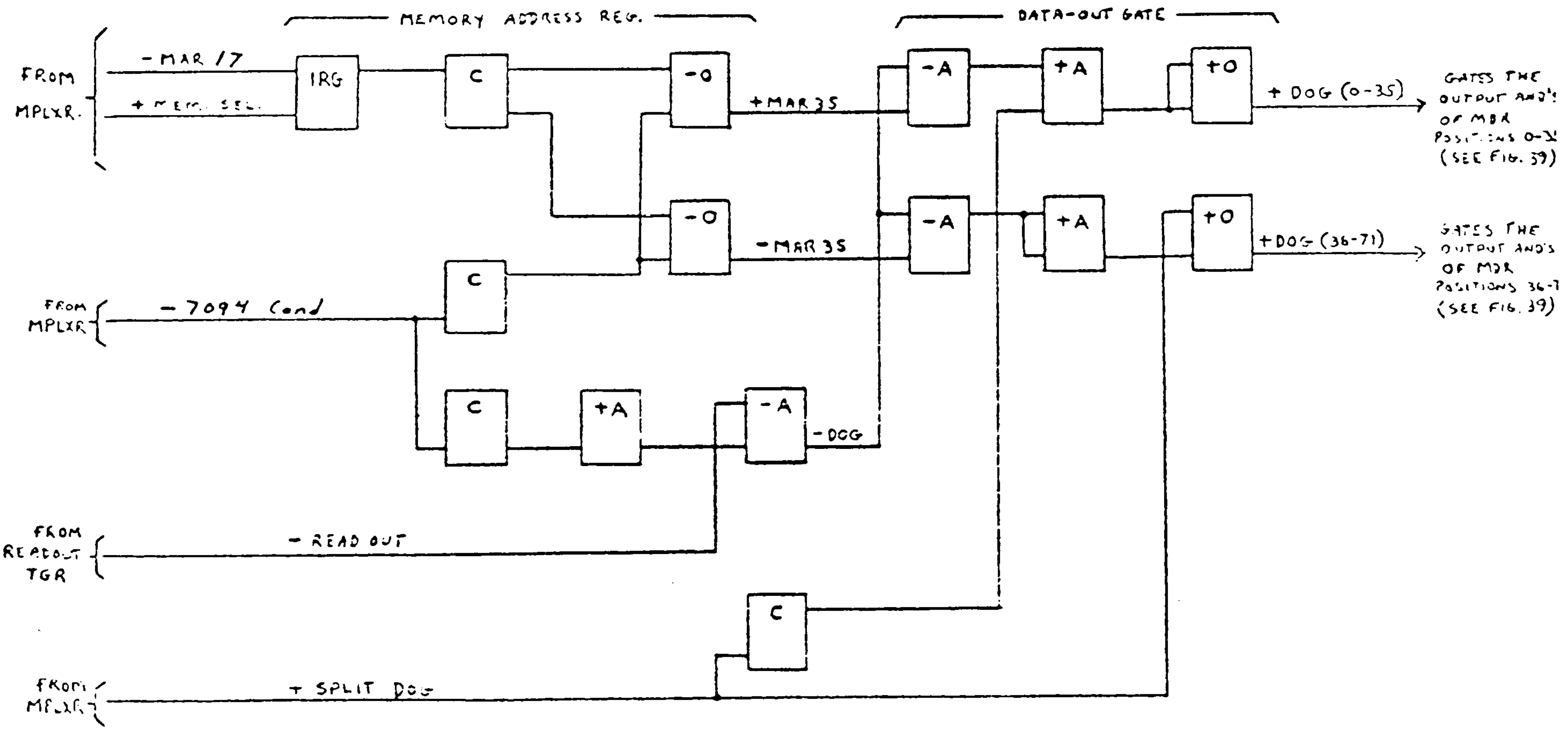
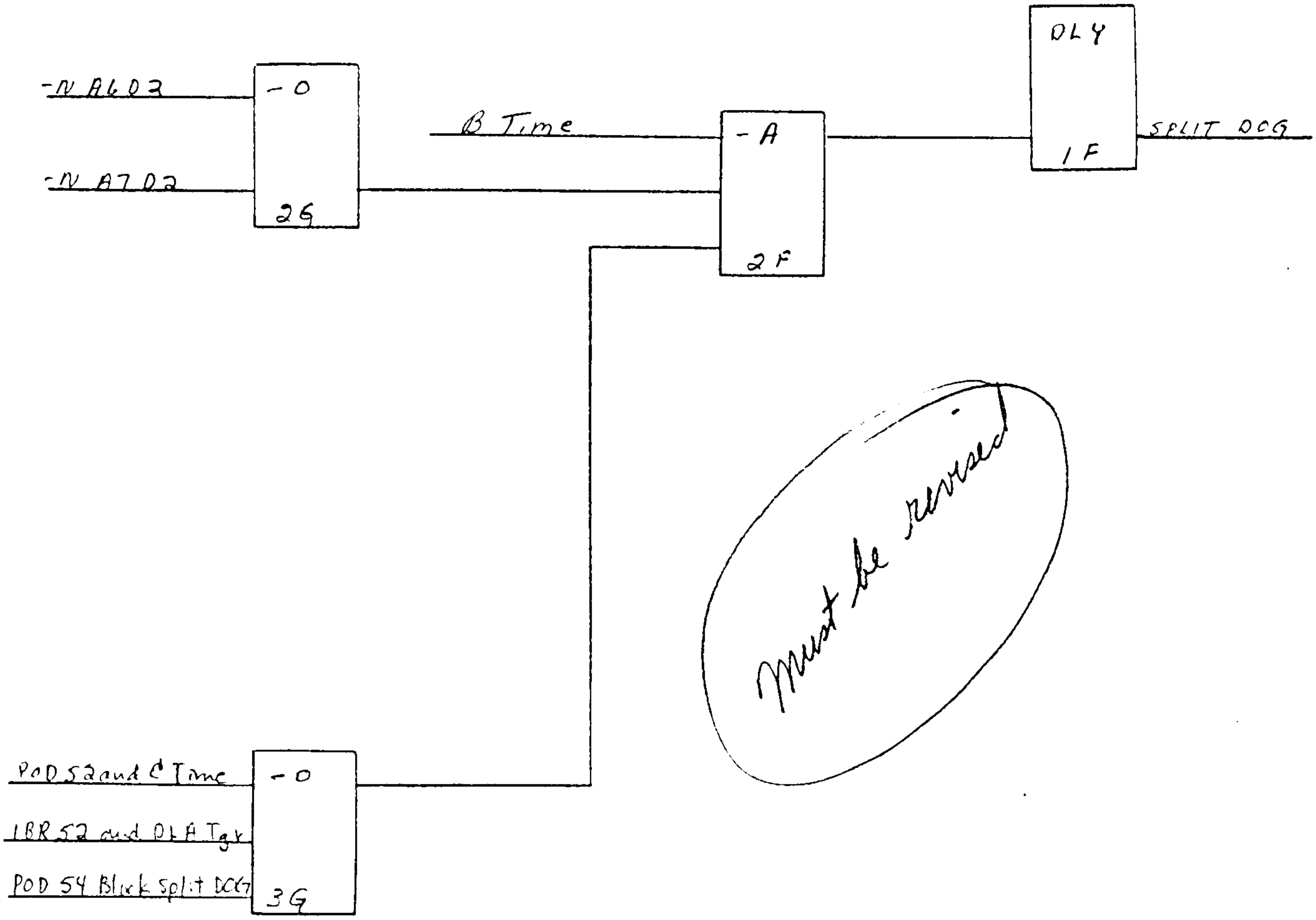


FIGURE 700 DATA-OUT GATING CIRCUITS FOR 7094 OPERATION



must be revised

Figure 701 Split DOG Control Circuits

IBR. This function is ~~directly~~ directly controlled by the load IBR trigger. Figure 702 shows the load IBR trigger will be set when :

1. AR 17 is not a 1.
2. P C 17 is not a 1.
3. It is A1(D1) time.
4. It is CPU 1 time.
5. The block load IBR trigger is not up.

With the load IBR trigger set, the SB is gated to the IBR at I9 delayed. The 70 nanosecond delay prevents gating out any of the even-address word. Note on Figure 702 that at I9 time the set IBR LDD (loaded) trigger is also set. Setting the IBR LDD trigger is one of the conditions necessary for enabling anyof the three kinds of instruction overlap operations. The conditions for setting and resetting the IBR LDD trigger are shown in Figure 703.

Up to this point the sequence is the same for every even-address core storage reference. After the second word is in the IBR and partial ~~deco~~ decoding completed, the sequence varies according to whether a DLA, TLA or SLA is going to be performed. The sequence concerning working with twc words during double precision arithmetic will not be considered in this section. It will suffice here to say that the condtiion of double precision operation is sensed and none of the instruction overlap control lines come up. The controlling factor here is I time -- instruction overlap is performed during I time while double precision operations are performed in E time -- therefore the computer must be in an E cycle in order to accept a double precision word.

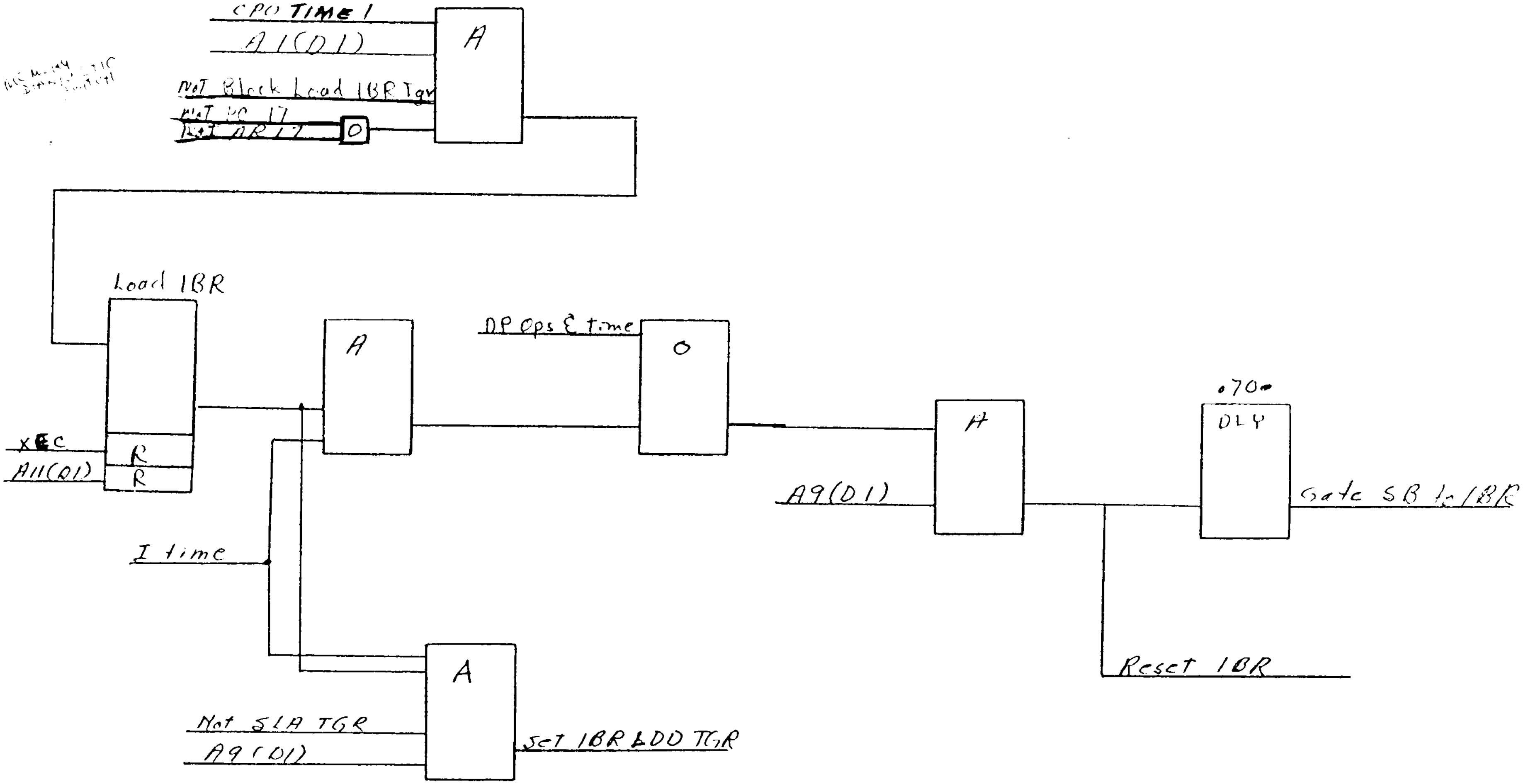
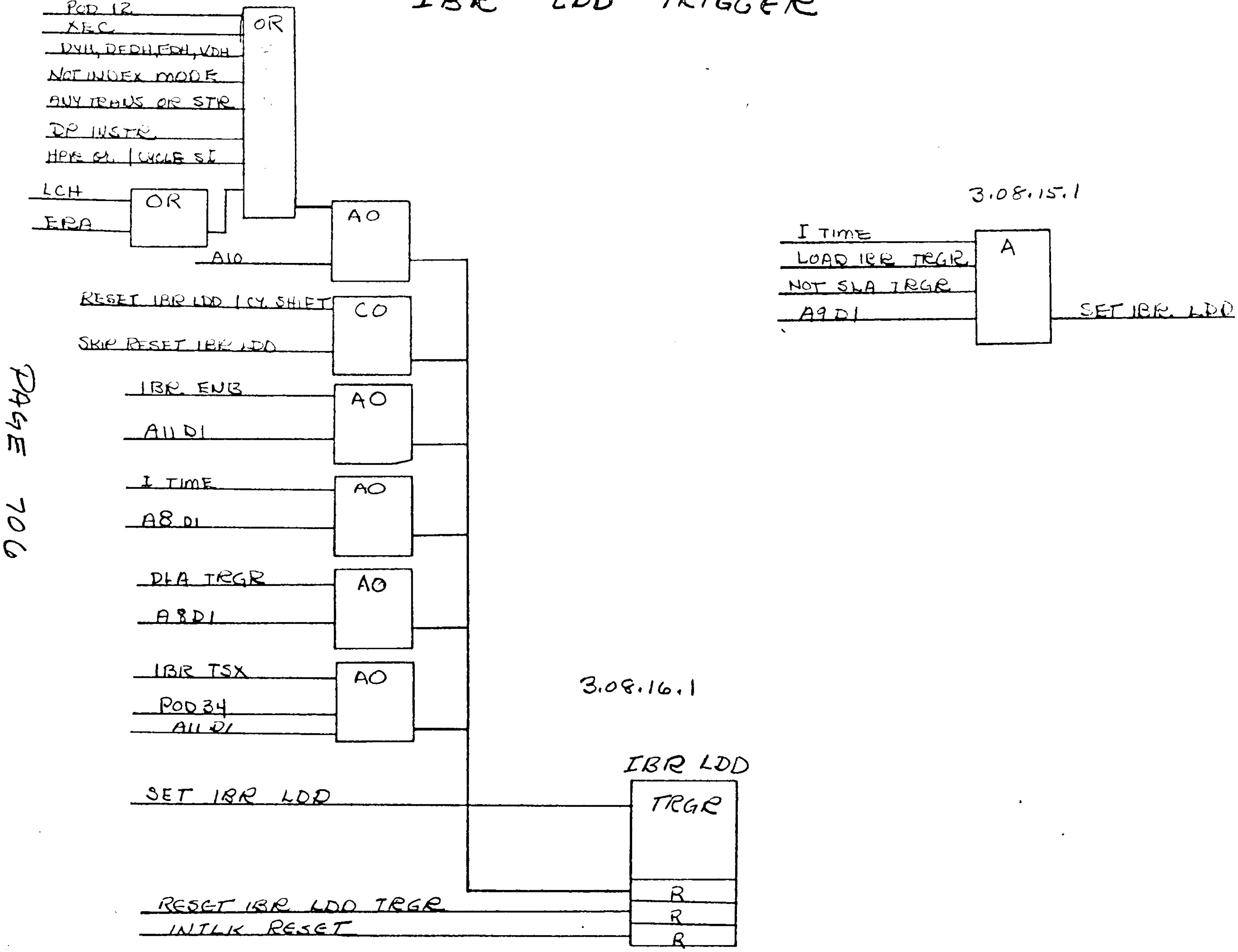


Figure 702, SB to IBR Gate and IBR Trigger

IBR LDD TRIGGER



701. Data Lookahead

The timing rules for data lookahead instructim overlap require that the next instruction in the even address have a cycle time of two or more cycles (excluding o6xx store type ~~xxxxk~~ codes) and that it be indexable. It is further required that the instnuction in the next higher odd address requires a data fetch (not a store operation) from core storage. When these conditions exist, the execution of the odd-address instruction is reduced by one cycle.

Figure 704 illustrates the sequence of events for a typical DLA instruction overlap. The instructions chosen for this illustration include a clear and add (CLA) instructikn at location 100, and an add (ADD)~~x,xxxxx~~ instruction at location 101. Without overlap these two instructions would require four machine cycles, and I and an E cycle for each instruction. With overlap it will be sbhown the total cycle time is r3duced from four to three. In the following discussion the paragraphs relate to the items called out on the illustration :

A. MX to SB -- the even addr4ss instruction from the multiplexor is available on the storage bus for two clock pulses, from 6 to 8 time. During this first I time the odd address instruction is also being gated out of memory to the storage bus and this is available from about 8 to 10d time. Although the split DOG control line comes up in the computer from 6 through 9 time, circuit delays result in its being felt in memory from 7 through 10 time.

B. SB to SR -- The even address instruction is gated from the storage bus to the storage register at I7. The [redacted] odd address

CLA-ADD showing lookahead

Item	Systems Pages	Activity	Remarks
A	01.00.01.1	MX to SB	
B	02.12.50.1	SB to SR	
C	02.01.00.1	SR	
D	03.04.04.1	Primary Op Decoder	
E	03.06.10.1	SR 21-35 to AR	
F	03.06.00.1	Address Register	
G	03.06.06.1	AR to XAD	
H	03.06.08.1	XAD to AR Rst	
I	03.06.08.1	Step PC	
J	03.06.30.1	Program Counter	Assume 100 CLA 101 ADD 102 Next Inst
K	03.05.15.1	AR to MAR	
L	03.00.07.1	PC to MAR	
M	03.00.16.1	E Time Call	
N	03.08.15.1	Set Load IBR Tgr	
O	03.08.15.1	Reset IBR	
P	03.08.15.1	Gate SB to IBR	
Q	03.08.15.1	Set IBR LDD Tgr	
R	03.08.15.1	IBR LDD Tgr	
S	03.00.19.1	MST E Time	
T	03.08.19.1	IBR to PR	
U	02.12.14.1	SR to AD	
V	02.12.31.1	AD to AC	End of CLA
W	01.00.09.1	End Op	
X	03.08.17.1	OLA Tgr On	Note this is the transition period between I6 and E7
Y	03.08.17.1	Change I to E time	
Z	03.06.10.1	IBR 21-35 to AR	
AA	03.06.08.1	AR to XAD	
BB	03.06.10.1	XAD to AR RsT	
CC	02.12.14.1	SR to AD	
DD	02.12.24.1	AC to AD	
EE	02.12.32.1	AD to AC	End of ADD
FF	03.06.10.1	IBR 18-20 to TR	Occurs simultaneous with Z

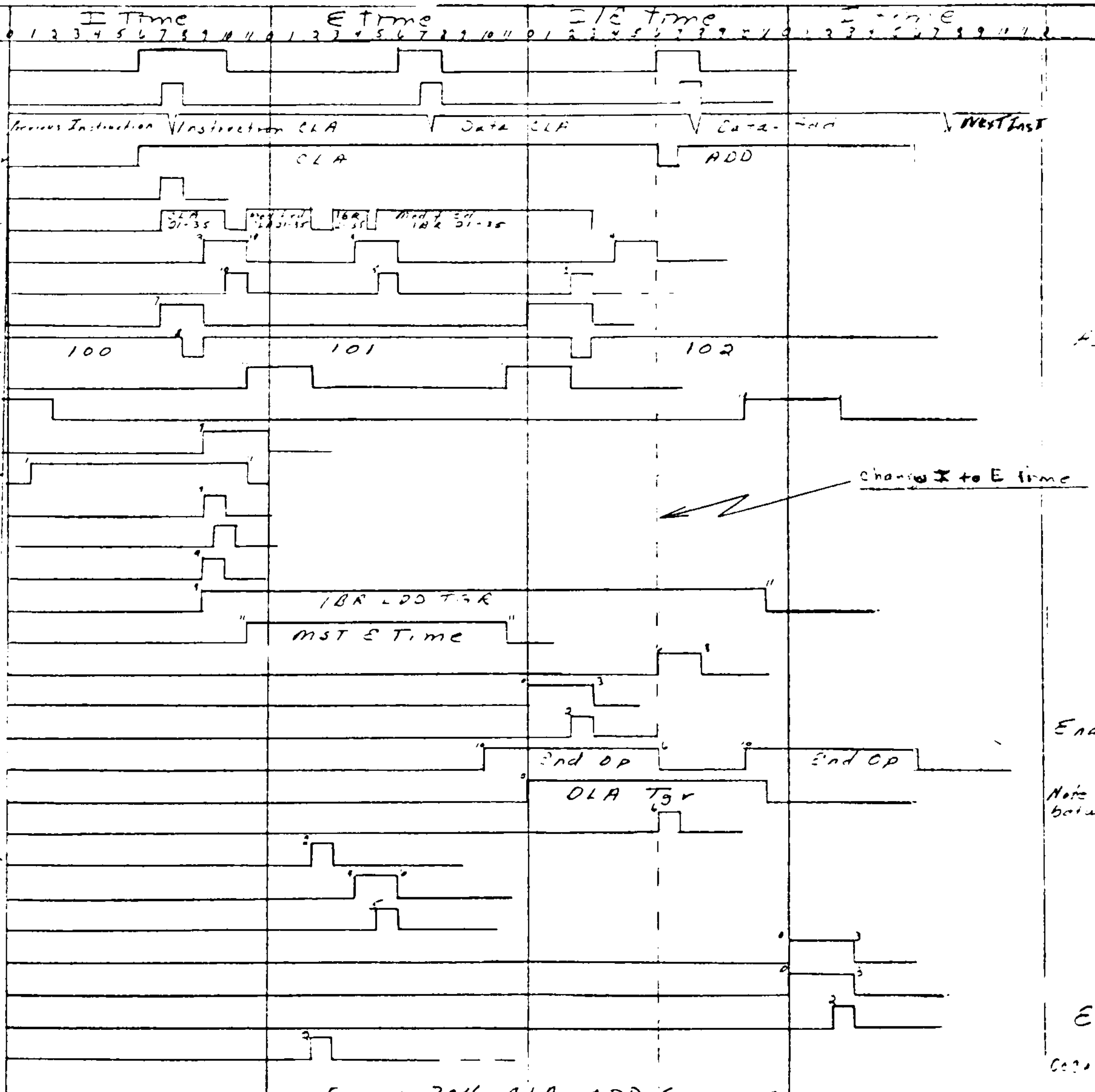


Figure 704. CLA-ADD Sequence

will be gated to the IBR at I9 time (see P).

C. SR -- at the beginning of the I cycle the storage register contained information relating to the previous instruction. Between I7 and I8 this is replaced with the CLA instruction. Between E7 and E8 the SR contains the data word (Operand) for the CLA instruction. At the next E8 time the SR contains the operand for the ADD instruction.

D. Primary Op Decoder -- from I6, through the E cycle, and to the following I6 time the primary operation decoder (program register) contains the CLA instruction. Control lines are conditioned to perform the clear and add instruction throughout this period. When the program register (PR) is reset, during the next I time, the contents of the IBR (S,1-9) are gated to the PR.

E. SR 21-35 to AR -- the SR contents, 21-35, are gated to the address register.

F. Address Register -- at this time the AR contains the unmodified core storage address of the CLA instruction .

G. AR to XAD -- at I9 the contents of the address register are gated to the index adders along with the contents of the specified index register (this last line is not shown).

H. XAD to AR Rst -- at I10 the contents of the index adders are taken back to the AR, resetting the AR to the modified address of the CLA instruction.

I and J. The program counter (PC) is stepped at I7 and again at I0 to contain the address of the ADD and the next instruction. The count of 101 is never used since this instruction was brought in with the instruction at location 100. Item L shows the PC is not gated to the memory address register (MAR) until the computer is ready for the instruction at location 102.

K. AR to MAR -- the contents of the address register are gated to MAR to bring the CLA operand out of memory. Note that the AR is again gated to MAR at E11 to bring out the operand of the ADD instruction.

L. PC to MAR -- the contents of the PC are gated to MAR for the instructions at locations 100 and 102. The contents of the program counter are not used when the count is 101.

NOTE: Whether the memory address register receives the contents of the PC or the AR is determined by the overlap conditions as shown in Figure 705.

M. E time Call -- E time call is initiated at I9.

N. Set Load IBR Trigger -- the set load IBR trigger control line is brough up at I1 and remains up through I11. The conditions necewsary to set this trigger are shown in Figure 702.

O. Reset IBR -- the IBR is reset at I9, preparatory to receiving the ADD instruction from the SB.

P. GAte SB to IBR -- at I9 (delayed) the contents of the SB are gated to the IBR.'

Q. Set IBR LDD Trigger -- at I9 the set IBR LDD Tgr control line comes up and sets the IBR LDD trigger. Conditions necessary to bring up this control line are shown in Figure 702.

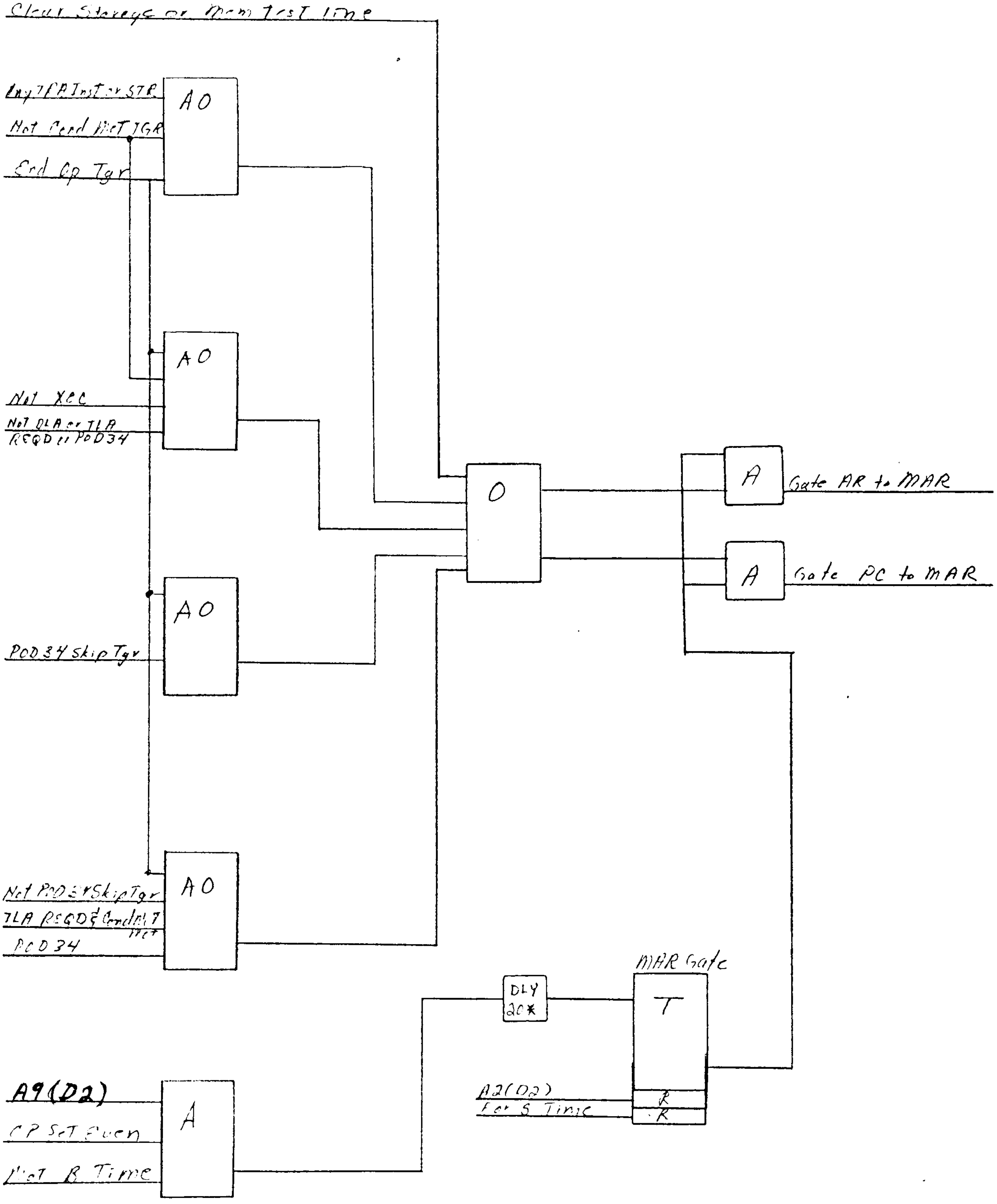


Figure 705. PC and AR Gating to MAR

R. IBR LDD Tgr -- the IBR LDD trigger is set at I9 and will remain on until the second E11 time. The IBR LDD trigger ;is one of the requisites for setting the necessary control circuitry to perform a data lookahead. The IBR LDD trigger is shown in Figure 703.

S. Mst E Time -- master E time comes up at I11 and remains up to E11. During this E cycle the ~~CLA~~ CLA operand will be brought from core storage (refer to items A and C).

T. IBR to PR -- the contents of IBR S, 1-9 are gated to the program register at E6 for decoding of the ADD instruction.

U. SR to AD -- during the preceding E cycle the CLA operand was brought to the storage register. Now, at I0, the contents of the SR are gated to the adders.

V. AD to AC -- At I2 the contents of the adders are gated to the accumulators and the CLA instruction is complete.

W. End Op -- at E10 the end operation trigger is turned on making it possible for the computer to go to an I cycle.

X. DLA Tgr ON -- at I0 the DLA trigger is turned on. Turn-on conditions necessary for the DLA tirgger are shown in Figure 706. This output is vital to computer operation during the transition at 6 time, i.e., when the computer is forced to go from I time to E time.

Y. Change I to E Time -- the change I to E time pulse resets (turns off) the master I time trigger (see Figure 306) and turns on the master E time trigger (Figure 311). By reverting to E time

DLM & SLM TGRS 03.06.11.1

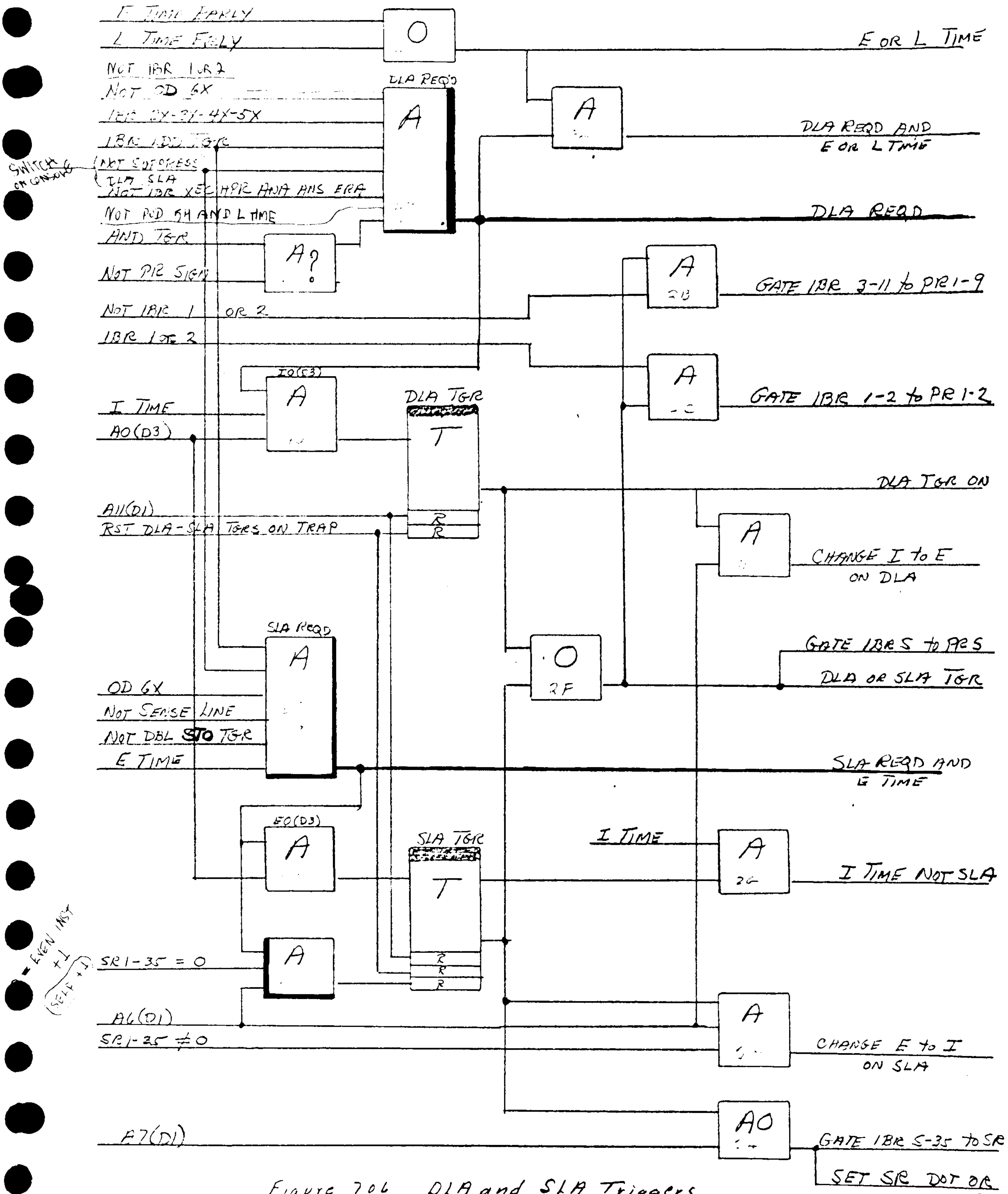


Figure 706 DLA and SLA Triggers

3027+

the computer can once again accept a data word from core storage (refer to item A). The I time portion of the cycle was necessary to complete the CLA instruction. E time is necessary to execute the ADD instruction.

Z, FF, AA and BB. This portion of the E cycle is similar to operations performed during I9 to I11, i. e., modifying the address portion of the ADD instruction.'

CC. SR to AD -- the contents of the storage register (the ADD operand) are gated to the adders at I0.

DD. AC to AD -- at I0 the contents of the accumulator (CLA operand) are also brought to the addərs and addition perfomeed.

EE. AD to AC -- at I2 the sum of the two operands is gated to the accumulator and the ~~conductor~~ computer will perform the next instruction.

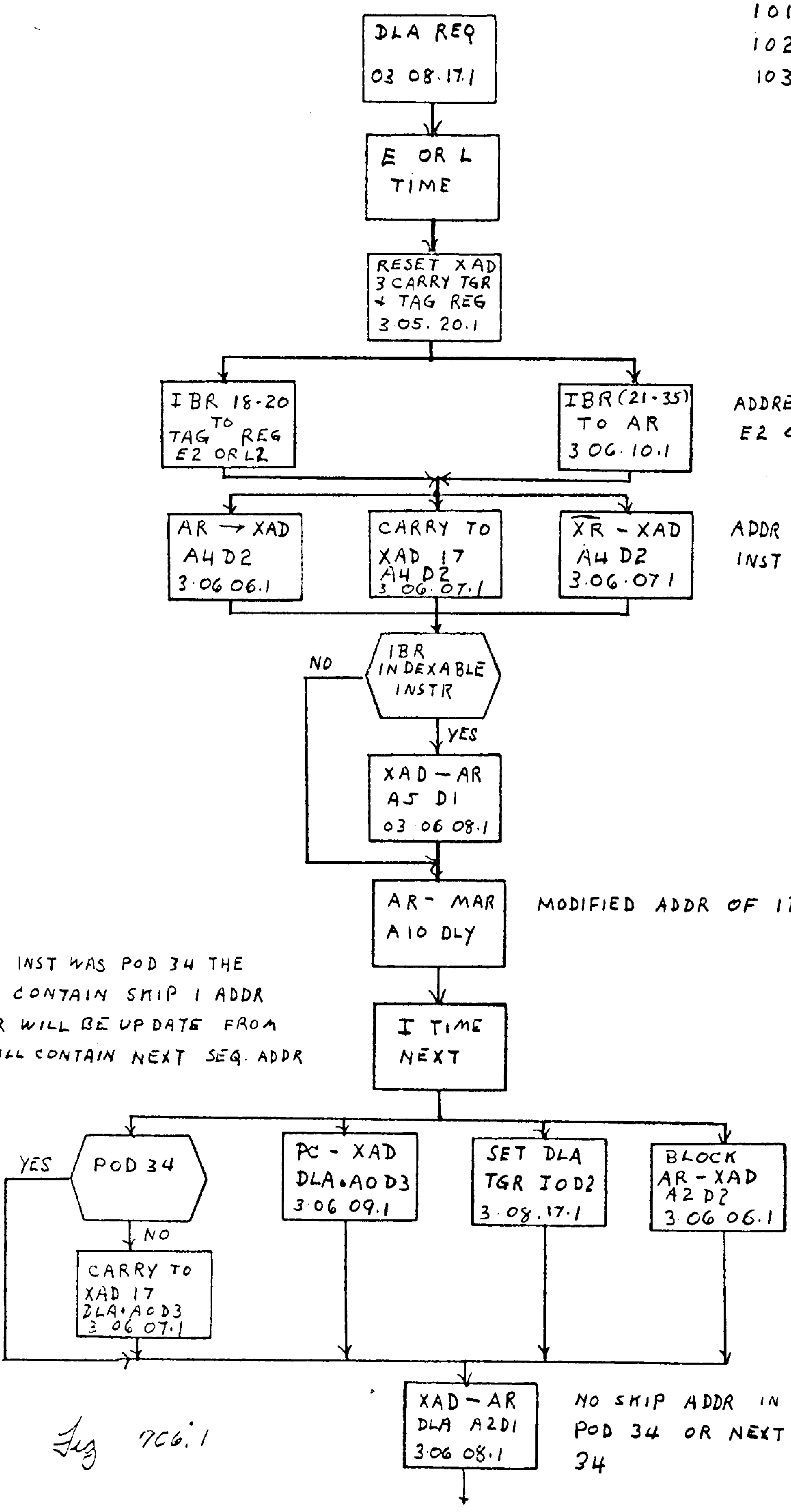
NOTE: The above procedure was ~~perform~~ performed in three cycles-- ah I, E and a composite I/E.

The following flow chart summarizes the activity during a data lookahead, beginning with the DLA request.

DATA LOOK AHEAD

EXAMPLE

100	CAS	600
101	CLA	700
102	SUB	670
103	TRA	300



ADDRESS PORTION OF CLA TO AR E2 OR L2

ADDR MODIFICATION OF INST IN IBR (CLA)

MODIFIED ADDR OF IBR INST TO MEMORY

IF EVEN INST WAS POD 34 THE PC WILL CONTAIN SKIP 1 ADDR (102) AR WILL BE UP DATE FROM PC + WILL CONTAIN NEXT SEQ. ADDR

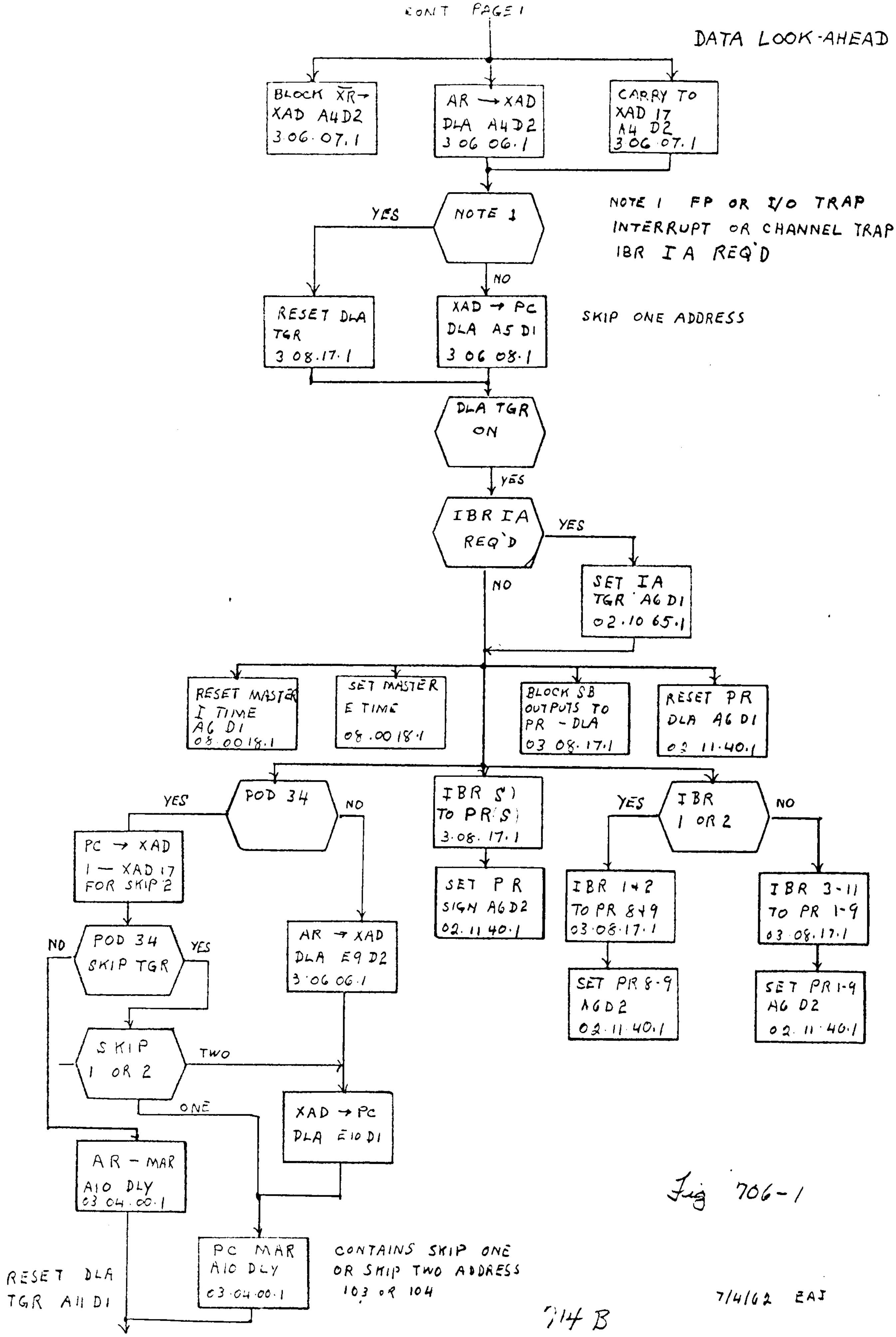
NO SKIP ADDR IN CASE ODD INST. IS A POD 34 OR NEXT SEQ INST IF NOT POD 34

Fig 706.1

CONT. PAGE 2

714A

DATA LOOK-AHEAD PAGE 2


CONT PAGE 1
BLOCK $\overline{XR}$ → XAD A4D2 3 06 07.1
AR → XAD DLA A4D2 3 06 06.1
CARRY TO XAD 17 A4 D2 3 06 07.1
NOTE 1
YES
NO
RESET DLA TGR 3 08.17.1
XAD → PC DLA A5 D1 3 06 08.1
DLA TGR ON
YES
IBR IA REQ'D
YES
NO
SET IA TGR A6 D1 02.10 65.1
RESET MASTER I TIME A6 D1 08.0018.1
SET MASTER E TIME 08.0018.1
BLOCK SB OUTPUTS TO PR - DLA 03 08.17.1
RESET PR DLA A6 D1 02 11.40.1
POD 34
YES
NO
IBR S1 TO PR S1 3.08.17.1
IBR 1 OR 2
YES
NO
PC → XAD 1 — XAD 17 FOR SKIP 2
SET PR SIGN A6D2 02.11 40.1
IBR 1+2 TO PR 8+9 03.08.17.1
IBR 3-11 TO PR 1-9 03.08.17.1
NO
POD 34 SKIP TGR
YES
AR → XAD DLA E9 D2 3 06 06.1
SET PR 8-9 A6D2 02.11 40.1
SET PR 1-9 A6 D2 02.11.40.1
SKIP 1 OR 2
TWO
ONE
XAD → PC DLA E10 D1
AR — MAR A10 DLY 03 04.00.1
PC MAR A10 DLY 03.04.00.1
RESET DLA TGR A11 D1

NOTE 1 FP OR I/O TRAP INTERRUPT OR CHANNEL TRAP DEMAND IBR IA REQ'D

SKIP ONE ADDRESS

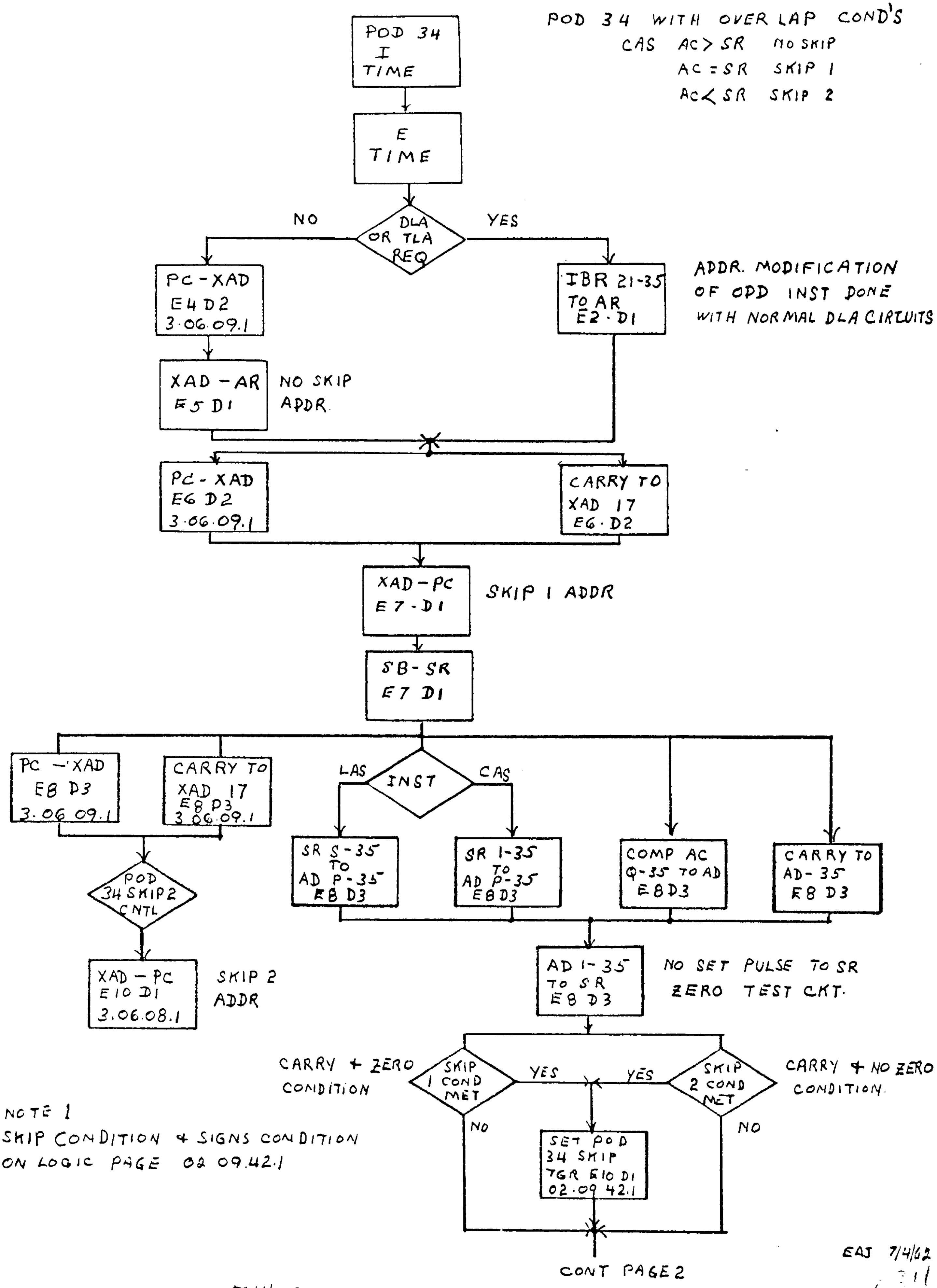
CONTAINS SKIP ONE OR SKIP TWO ADDRESS 103 OR 104

Fig 706-1

7/4/62 EAJ

714 B

POD 34 WITH OVER LAP COND'S
CAS AC > SR NO SKIP
AC = SR SKIP 1
AC < SR SKIP 2



ADDR. MODIFICATION OF OPD INST DONE WITH NORMAL DLA CIRCUITS

NO SET PULSE TO SR ZERO TEST CKT.

CARRY + ZERO CONDITION

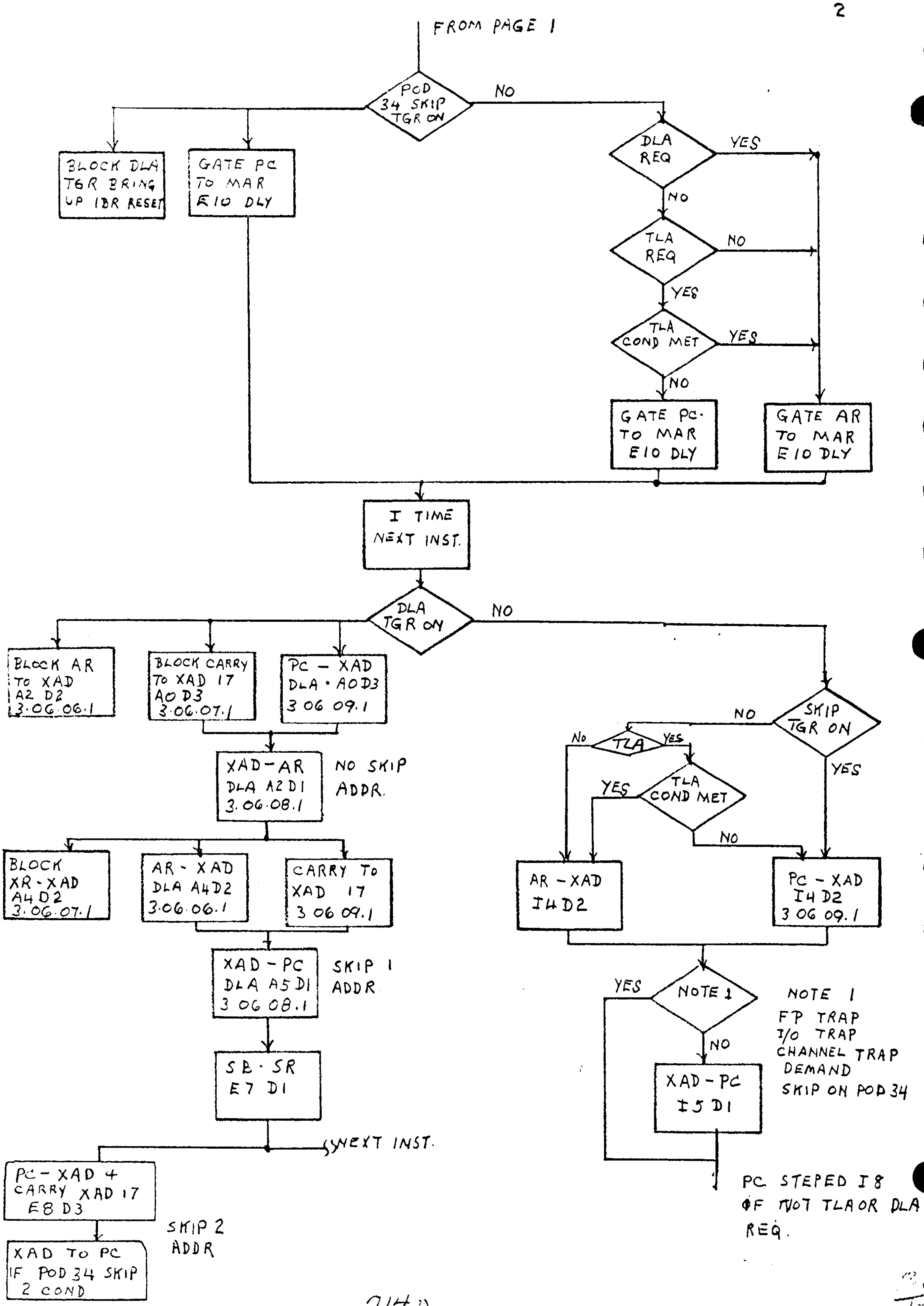
CARRY + NO ZERO CONDITION.

NOTE 1
SKIP CONDITION + SIGNS CONDITION
ON LOGIC PAGE 02 09.42.1

EAJ 7/4/62

311

714 C



7143

7/4/62 EAJ

702. Transfer Lookahead

The instructions ADD-TIX have been selected to show the sequence of a transfer lookahead. A transfer lookahead may be performed if the instruction in the even location has an octal code of 02xx, 03xx, 04xx, or 05xx (except 052x and 053x) and the next instruction is TRA, TTR, TXI, TIX, TNX, TXH, TXL, or TSX. When these conditions exist, the odd address instruction is executed in overlapped condition and requires no cycle time.

ADD-TIX is a common loop in programming. Assume we are using the two instructions to add successive locations in memory, building up a composite sum in the accumulator. At the end of the loop ~~the~~ the sum may either be stored or used in the performance of the next instruction.'

In studying the following sequence notice that the TIX instruction never actually appears -- although it is performed. The sequence refers to Figure 707. Alphabetical notations on the paragraphs are keyed to the sequence chart.

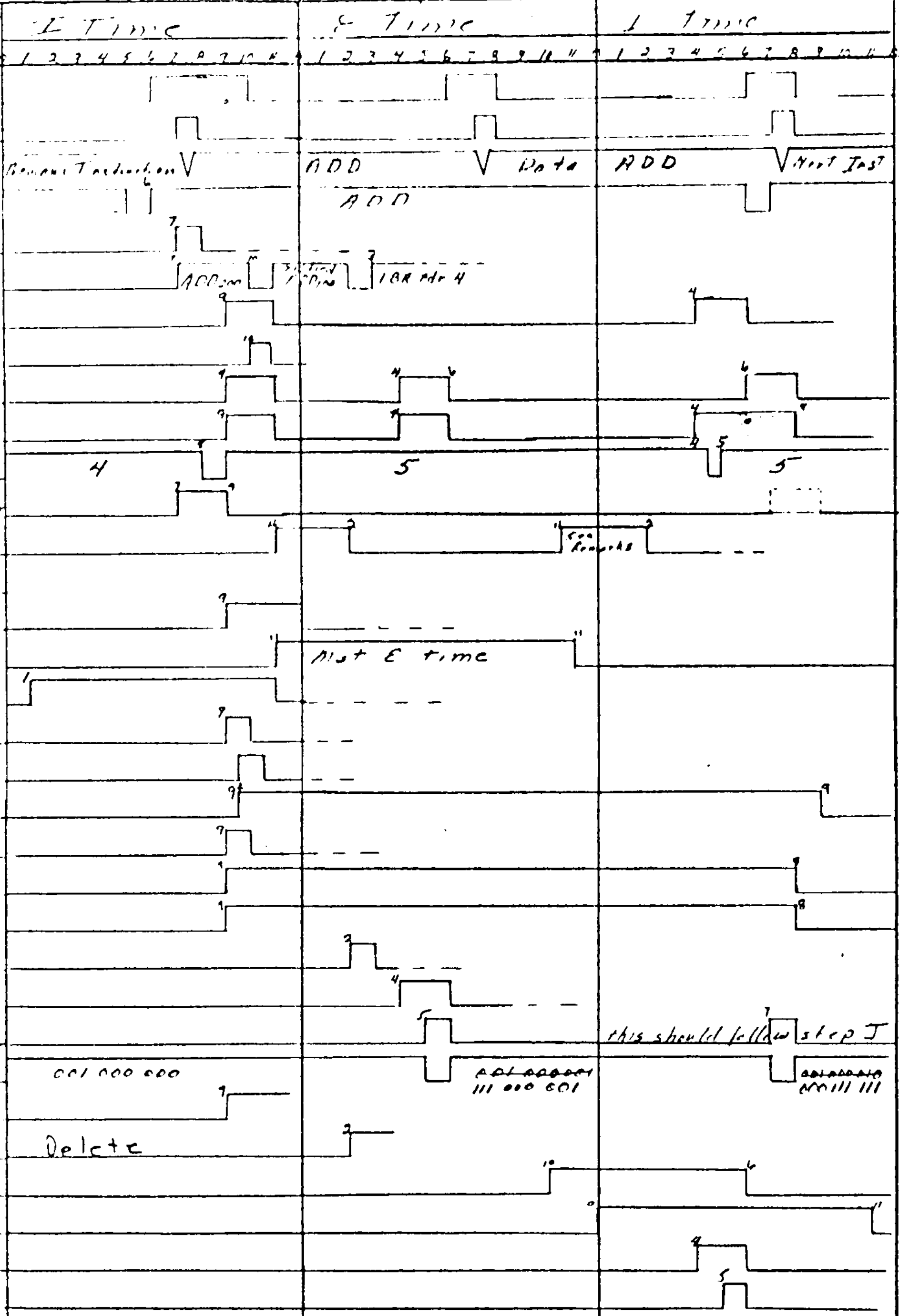
A. MX to SB -- the even address instruction is gated from the multiplexor to the storage bus at I6 to I8. The odd address, the TIX instruction, will be on the storage bus from I8 to about I10.

B. SB to SR -- an I7 (D1) pulse gates the information on the storage bus into the storage register.

C. Storage Register -- until I7 the storage register will contain information relating to the previous instruction. Between I7 and I8 the contents of the storage register are reset to the ADD

ADD-TIX Showing Transfer Lookahead

Item	Systems Page	Activity
A	03.00.01.1	MX to SB
B	03.10.01.1	SB to SR
C	03.04.01.1	Storage Register
D	03.04.07.1	Program Register
E	03.06.10.1	SR21-35 to AR
F	03.06.01.1	Address Register
G	03.06.06.1	AR to XAD
H	03.06.08.1	XAD to AR Reset
I	03.06.07.1	XR to XAD
J	03.06.07.1	1 to XAD 17
K	03.06.30.1	Program Counter
L	03.06.08.1	Step Program Counter
M	03.08.15.1	AR to MAR
N	03.00.07.1	PC to MAR
O	03.00.16.1	E Time Call
P	03.00.19.1	Master E Time
Q	03.08.15.1	Set Load IBR Tgr
R	03.08.15.1	Reset IBR
S	03.08.15.1	Gate SB to IBR
T	03.08.19.1	IBR TIX
U	03.08.15.1	Set IBR LDD Tgr
V	03.08.15.1	IBR LDD Tgr
W	03.08.19.1	TLA Request
X	03.05.20.1	Reset XAD 3 Carry Tgr
Y	03.06.13.1	IBR(3-17) to XAD
Z	03.12.20.1	XAD to XR
AA	03.05.30.1	XR2
BB	03.06.19.1	TLA cond met if 3 Tgr off
CC	03.06.19.1	TLA cond met if Tgr on
DD	03.00.09.1	End Op
EE	03.08.18.1	TLA Tgr ON
FF	03.06.07.1	Block XR → XAD
GG	03.06.08.1	XAD to PC



Remarks

Program:
3) TXT 100,2
4) ADD 200,2
5) TIX 1-1,5,1

step PC blocked by TLA TGR 03.06.09.1

uses E10 delayed

Figure 707
p 716

716

instruction. Between E7 and E8 the storage register will be reset to the operand of the ADD instruction.

D. Program Register -- while the two instructions ADD-TIX are being performed the program register will contain only the ADD instruction for decoding.

E. SR 21-35 to AR -- contents of the SR 21-35 are gated to the address register at I7. This is the unmodified address of the ADD instruction.

F. Address Register -- from I7 through I10 the address register contains the unmodified address of the ADD instruction. The address register will be set at I11 to contain the modified address of the instruction. The following E3 the address register is again reset to the address specified by the address portion of the instruction in the IBR.

G. AR to XAD -- the contents of the Ar are gated to the index adders at I9 for address modification. Steps G, H, I and J comprise the ~~mxxfixdx~~ modification sequence.

H. XAD to AR Rst -- at I10 the contents of the XAD are gated back to the AR (this is following address modification).

I. XR to XAD - the complement of the contents of index register two are brought to the XAD at I9 to modify the address coming in from the address register.

J. 1 to XAD 17 -- adding a 1 to XAD position 17 ~~xhxxnx~~ changes the contents of the XAD to the 2's complement of the index register.

K. Program Counter -- the contents of the program counter are not used during this overlap sequence. However, the program counter is stepped correctly until the second I time when the step pulse is blocked and the program counter is reset to a count of 5. This would be the corr~~x~~ect count if the prgram counter were to be gated to MAR.

L. Step Program Counter -- refer to step K.

M. AR to MAR --at I11 the modified address in the AR is gated to MAR. This address will locate~~d~~ the ADD operand in memory. At the next E11 the modified address of the TIX instruction is gated to MAR by an E10(delyaed) pulse.

N. PC to MAR -- does not occur.

O. E Time Call -- the E time call is initiated at I9.

P. Master E Time --master E time begin s at I11 and remains up for a full cycle.

Q. Set Load IBR Trigger -- the set load IBR trigger comes up at I1 and remains up to I11. The conditions necessary to set this trigger are shown in Figure 702.

R. Reset IBR -- the IBR is reset at I9, preparatory to receiving the TIX instruction.

S. GAte SB to IBR -- at I9 (delayed) the contetnts of the SB are gated to the IBR.

T. IBR TIX -- at I9, as soon as the TIX instruction is in the IBR, the instruction is decoded and control lines come up preparing the computer for the transfer instruction.

U. Set IBR LDD Trigger -- at I9 the set IBR LDD trigger control line comes up and sets the IBR LDD trigger. Conditions necessary to bring up this control line are shown in Figure 702.

V. IBR LDD Trigger -- the IBR LDD trigger is set at I9 and will remain up until the next I8. This trigger must be set to perform a transfer lookahead.

W. TLA Request -- the TLA request is conditioned by the IBR LDD trigger and remains up for the same period. Conditions necessary' for a TLA request are shown in Figure 708.

X. Reset XAD 3 Carry trigger -- this trigger is reset at E2 if the 1 injected into XAD 17 and the addition of the XR to the AR results in a carry out of the XAD position. This would occur, for example, when the XR and AR were equal, signifying the ADD-TIX loop had been completed. When the 3 carry trigger is up the TLA request circuitry is nullified.

Y. IBR (3 - 17) to XAD -- this gates the decrement of the TIX instruction to the XAD for modification of the XR.

Z. XAD to XR -- note in step I the KR was brought to the XAD at E4, simultaneously with the contents of IBR 3-17. The modified contents of the XAD are now returned to the index register.

AA. This line shows the contents of the index register throughout the cycle. Recall that the complement of the register is always used in the index adders.

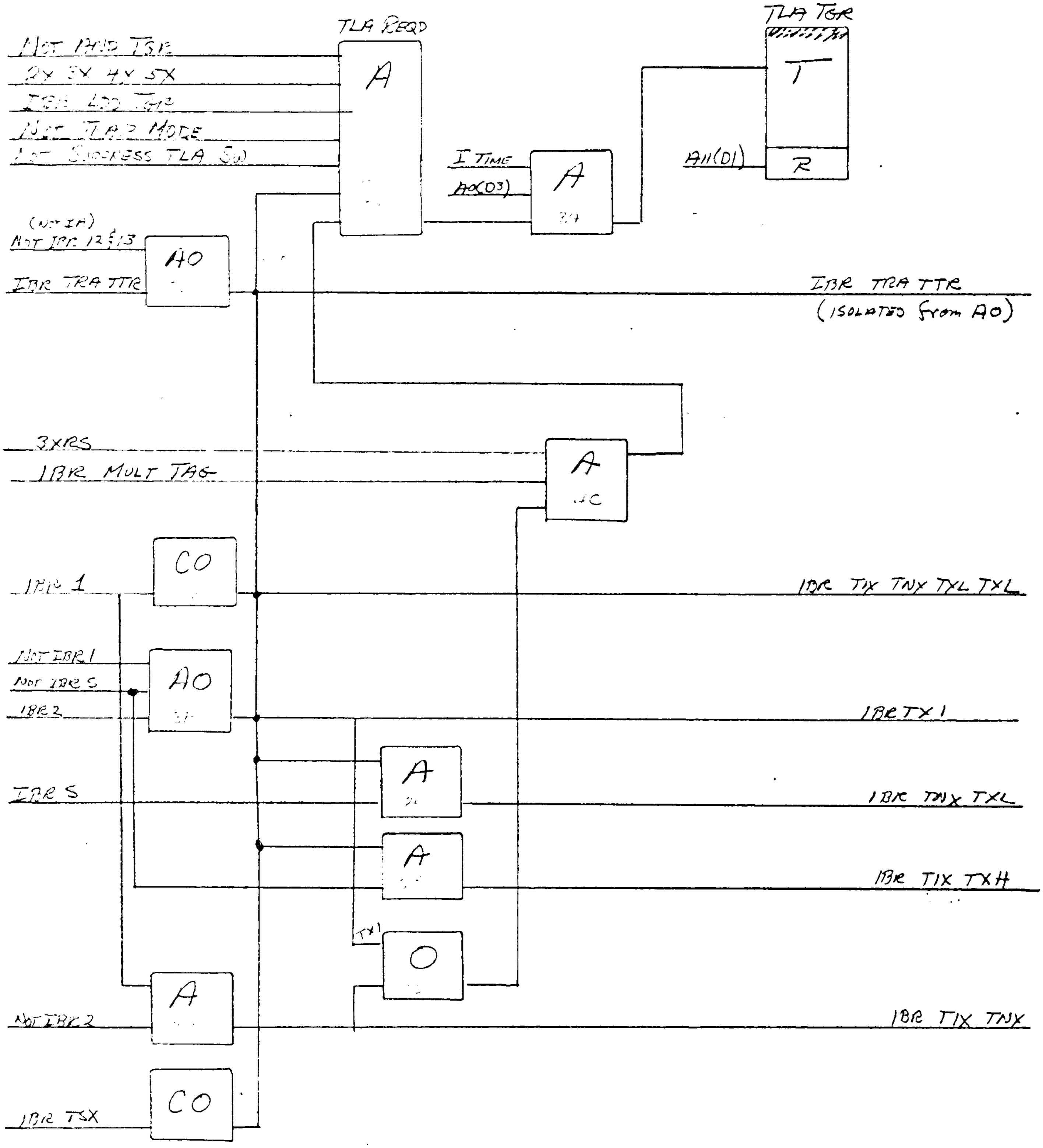


Figure 708 TLA Request and TLA Trigger

BB. TLA Condition Met -- this line will come up at I9 if a transfer is to take place. The two conditions necessary to bring up this control are a TIX instruction and no carry out of XAD 3.

CC. Delete.

DD. End Op -- the end operation trigger is turned on at E10, enabling the computer to go to an I cycle.

EE. TLA Tgr ON -- the TLA trigger is turned on at I0 and remains on until I11. Conditions necessary to set the TLA trigger are shown in Figure 708. When this trigger is on the step program counter pulse is blocked -- refer to item K.

FF. Block XR to XAD -- the XR woubd normally be gated to the XAD at I4. Notice, however, that in step G we are going to modify the address register from E4 to E6.

GG. XAD to PC -- at E5 the contents of the XAD are gated to the program counter. Although this step seems unnecessary in this sequence, certain skip instructions take advantage of modifying the program counter in this fashion. The program counter now contains the TIX address count plus 1 (in step J the 1 to XAD 17 is held up over a count of four clock pulses).

This ADD-TIX loop will continue until the count in the index register and address register are equal. At this point the program will drop through to the next instruction.

The flow diagrams on the following pages relate to various transfer instructions in the IBR, i. e., transfer overlap instructions.

IBR TIX TNX TXH, TXL

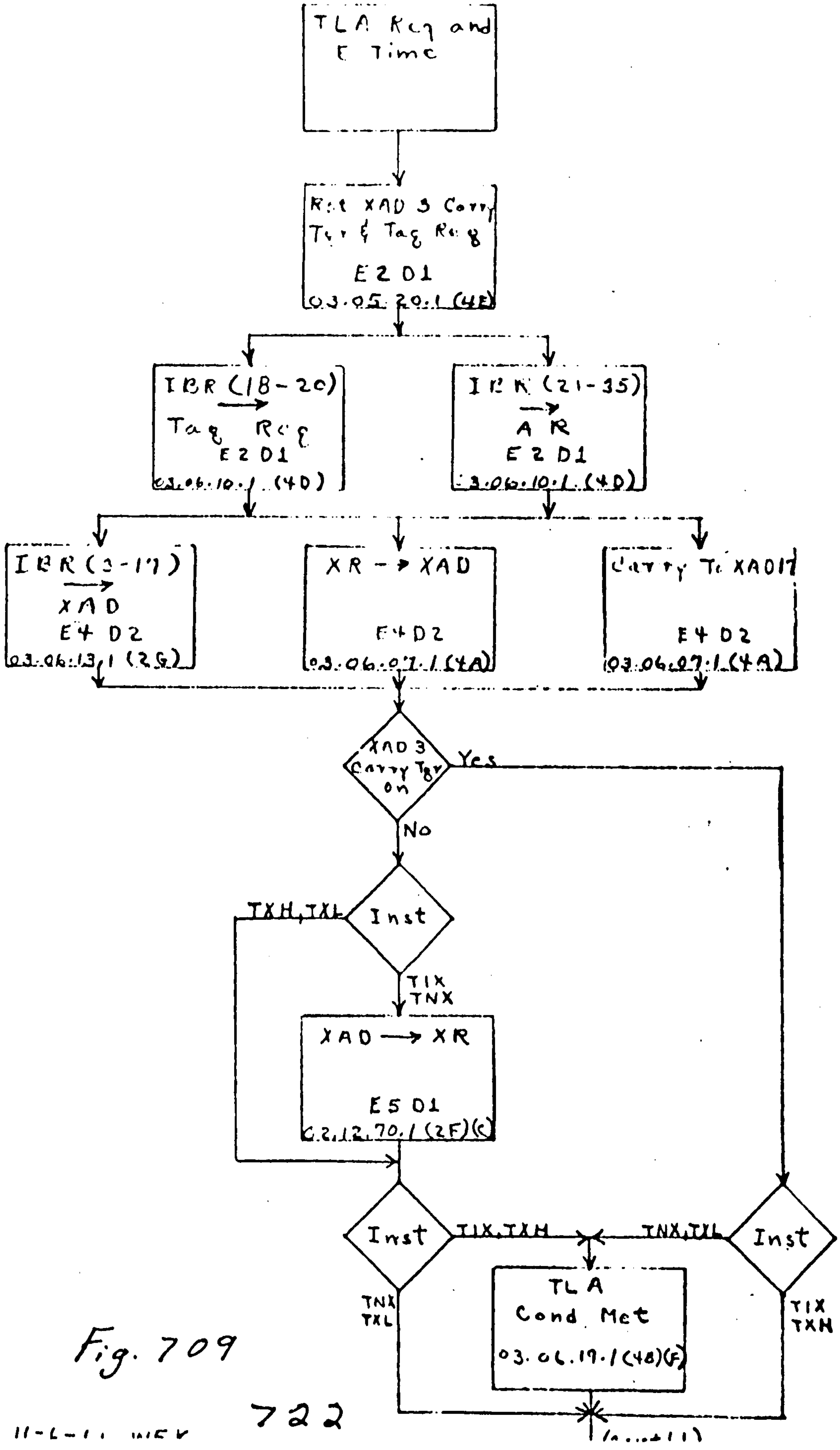
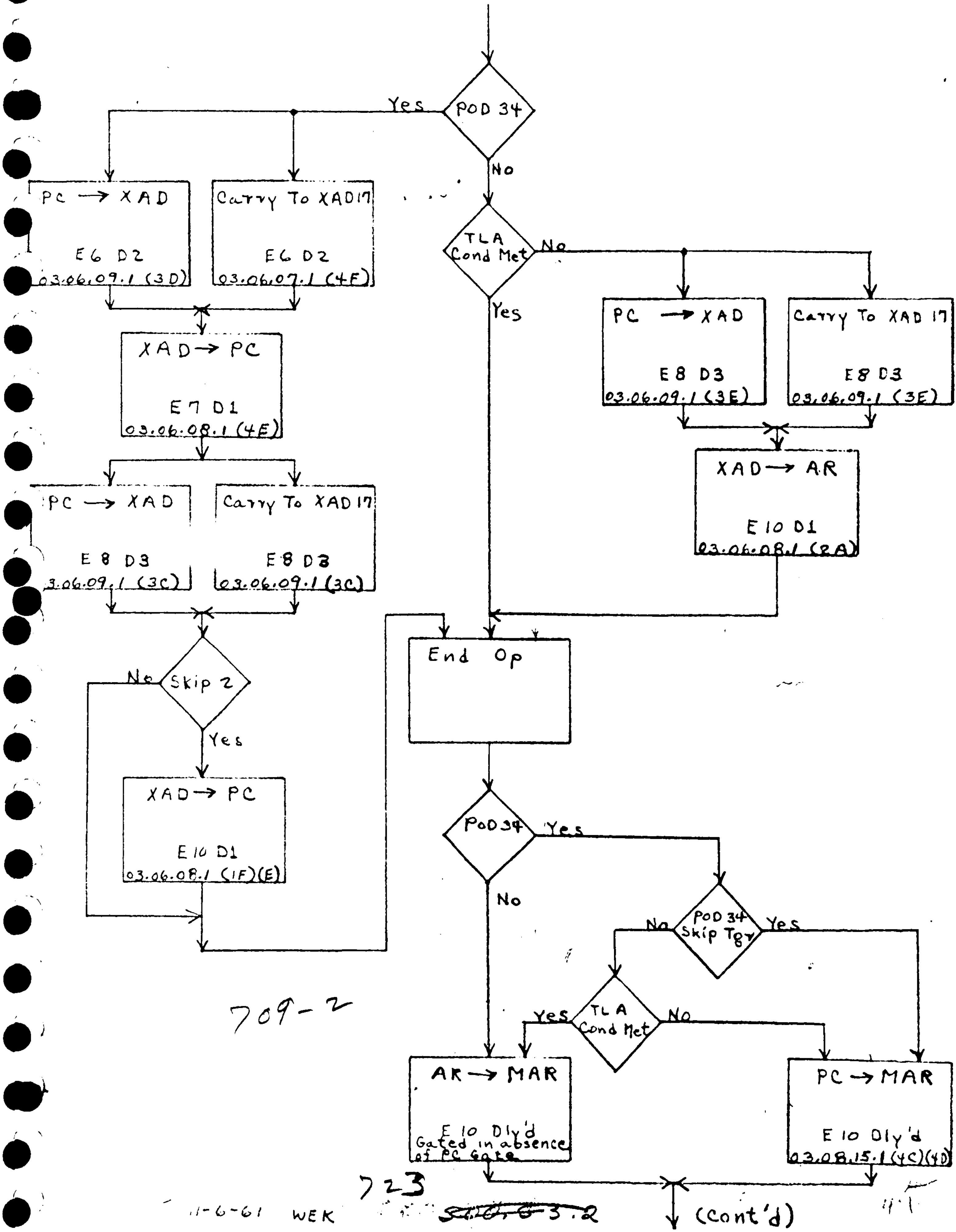


Fig. 709

11-6-61 WEK

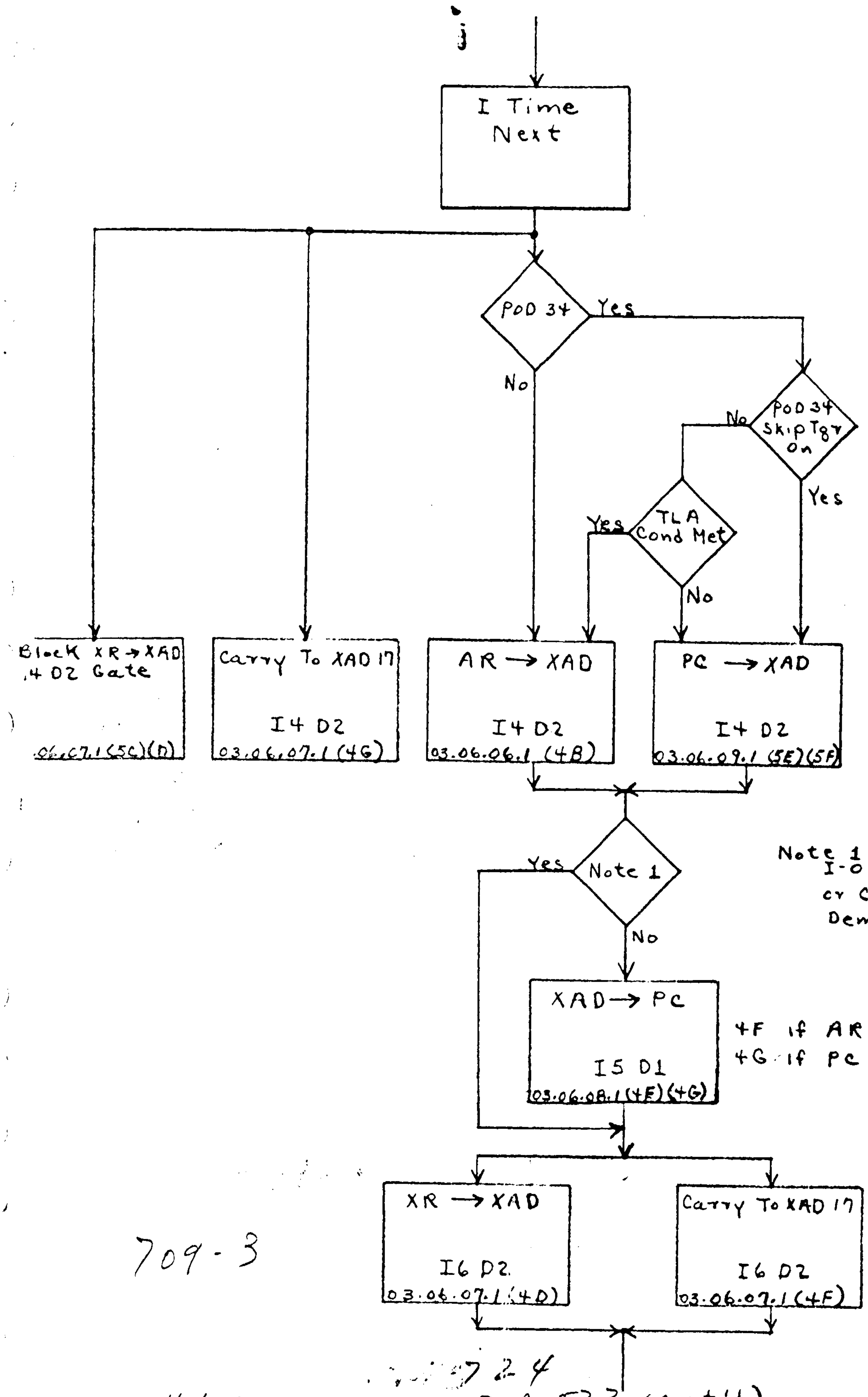


709-2

723

11-6-61 WEK ~~5.1.6.6~~ 3.2

IBR TIX, TNX, TXH, TXL (cont'd) (2)



Note 1 = FP Trap or I-O Trap or Interrupt or Channel Trap Demand

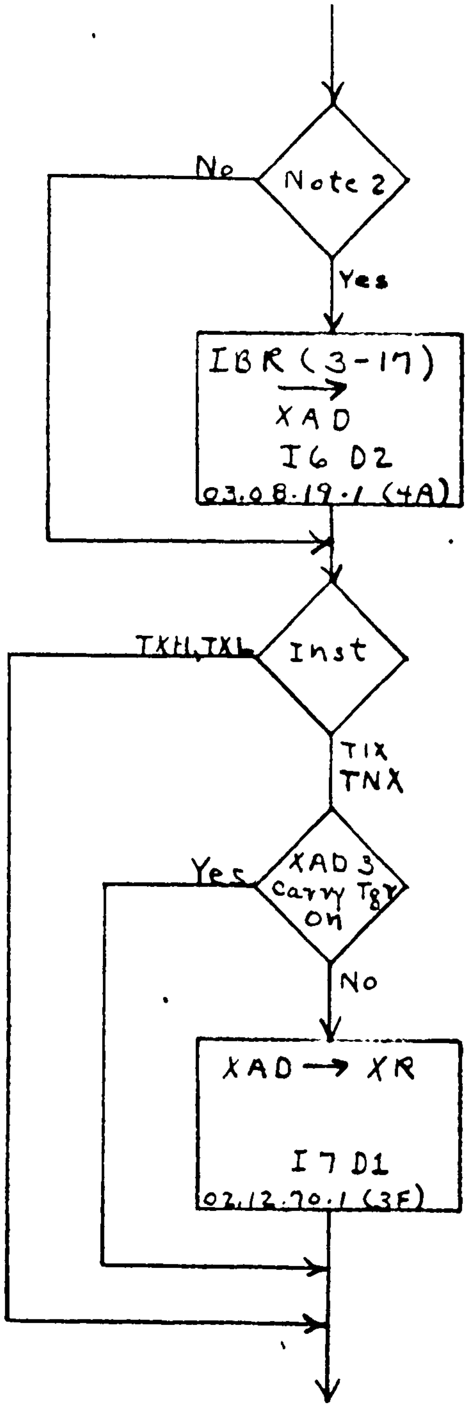
4F if AR → XAD
4G if PC → XAD

709-3

724

(cont'd)

11-6-61 WEK



Note 2 = FP Trap or I-O Tra
or Interrupt or Channel
Trap Demand or POD 34
Skip Tgr

709-4

725

~~500.53.4~~

11-6-61 WEK

IBR TXI

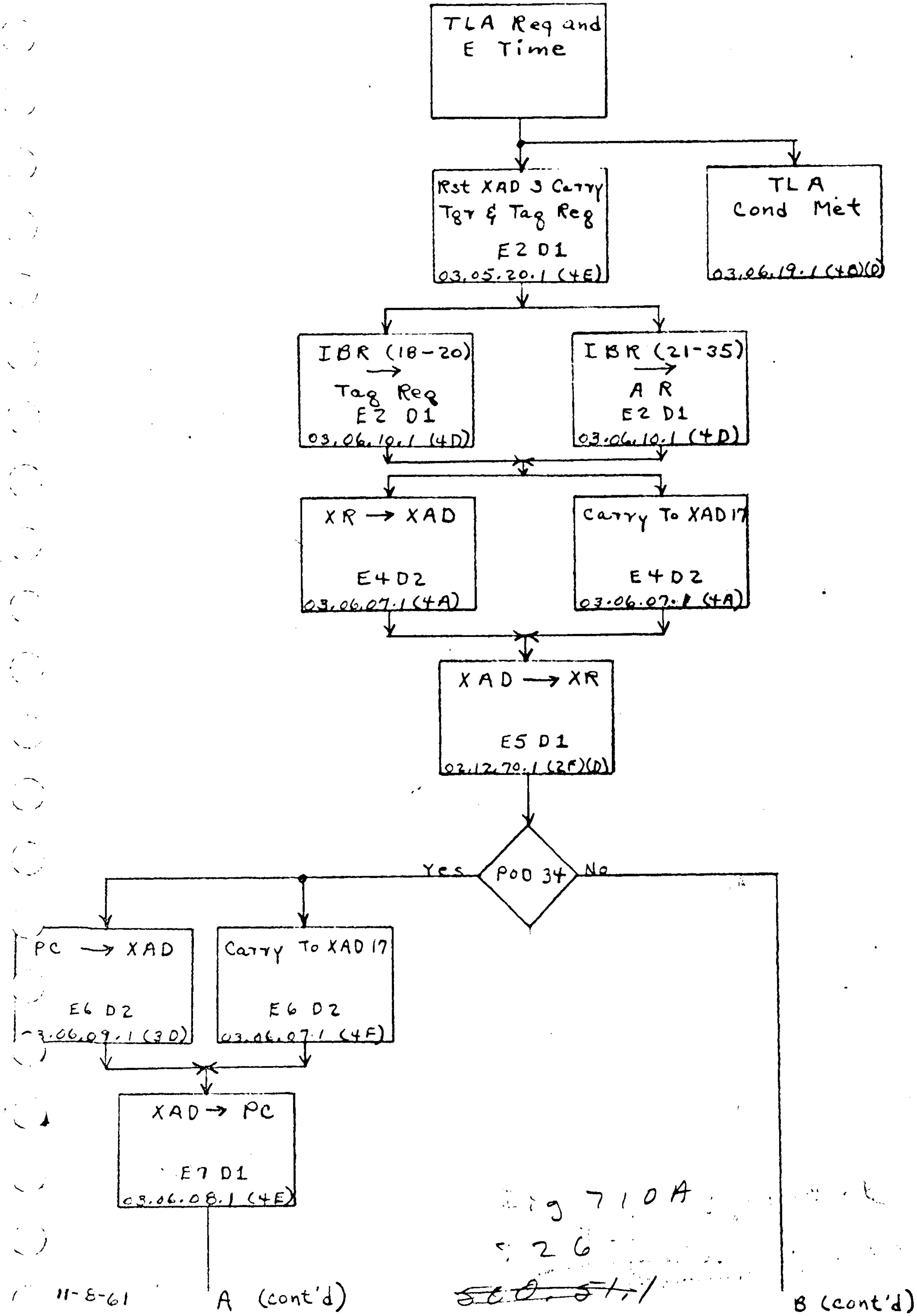


Fig 710A

26

~~5.0.5.1.1~~

11-8-61

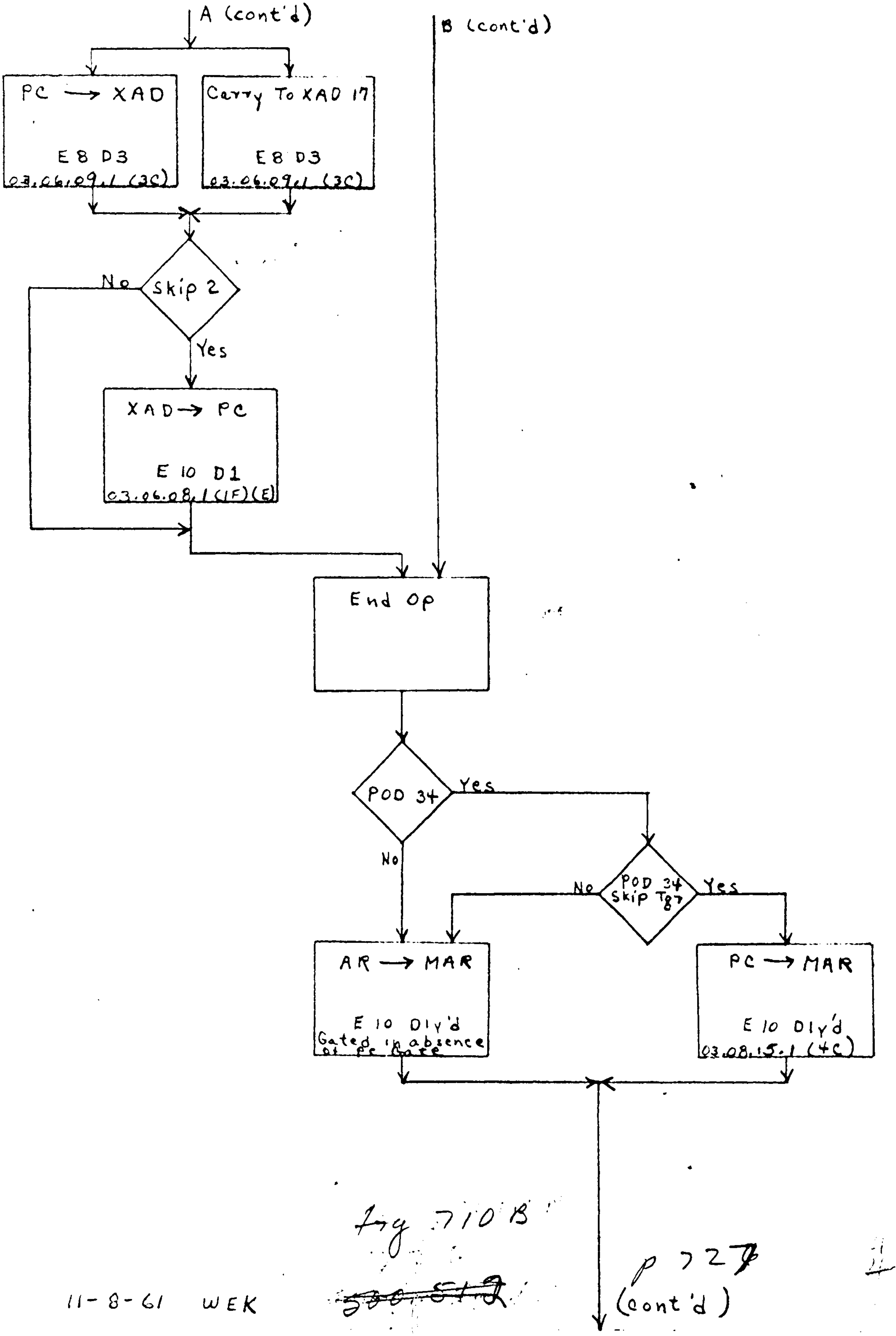
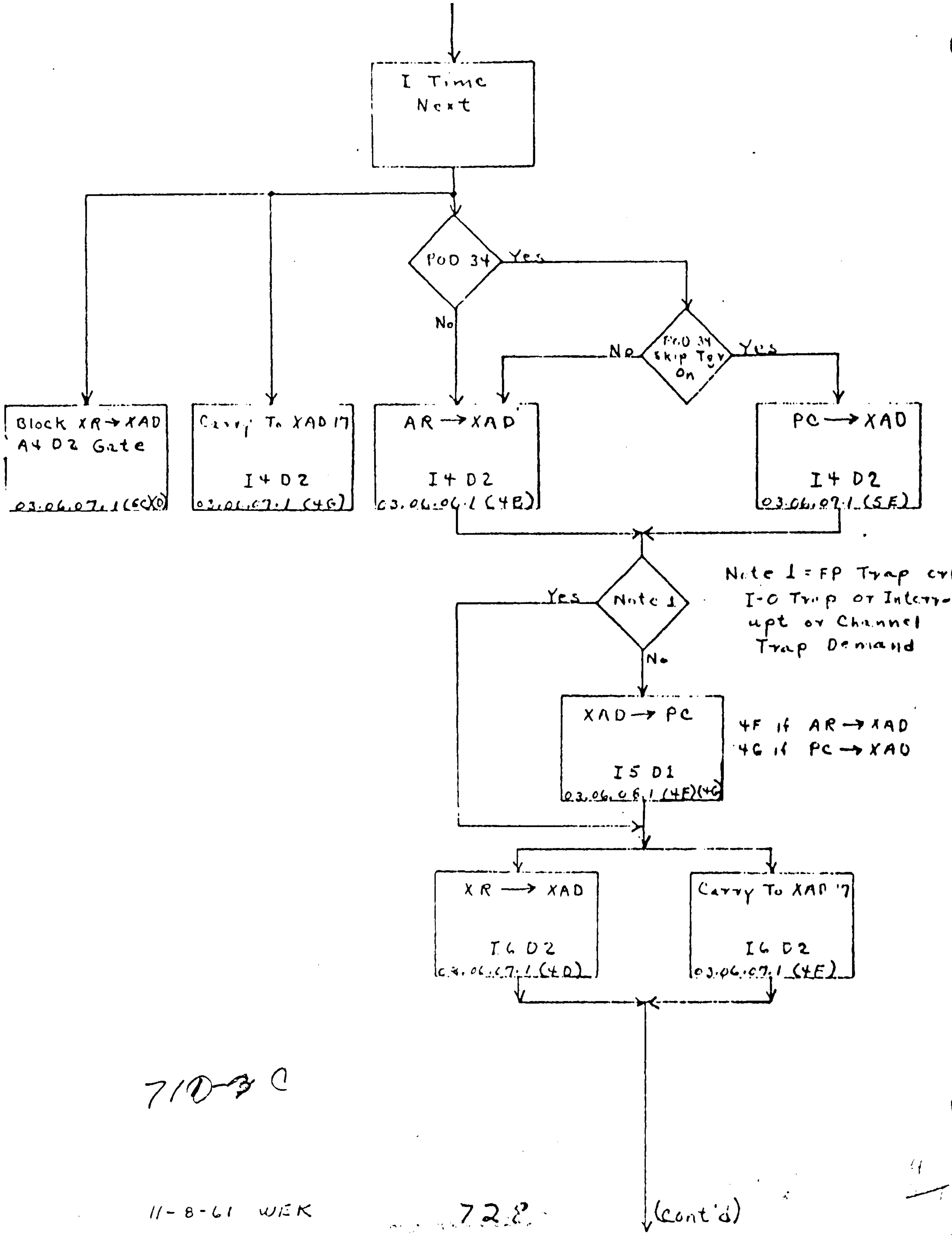


fig 710B

11-8-61 WEK ~~530 512~~

IER TXI (cont'd) (2)



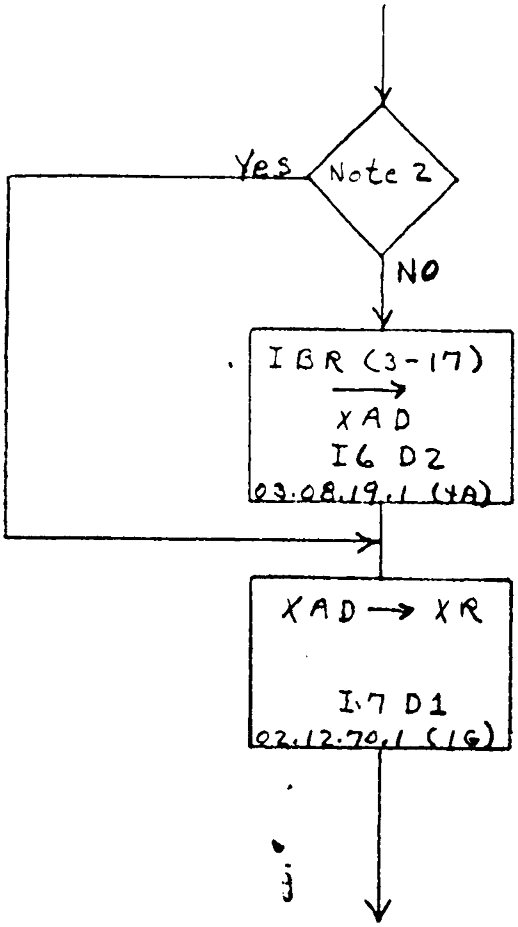
Note 1 = FP Trap or I-O Trap or Interrupt or Channel Trap Demand

4F if AR → XAD
4G if PC → XAD

710-3 C

11-8-61 WEK

728



Note 2 : FP Trap or I-O Trap or Interrupt or Channel Trap Demand or POD 34 Skip Tgr

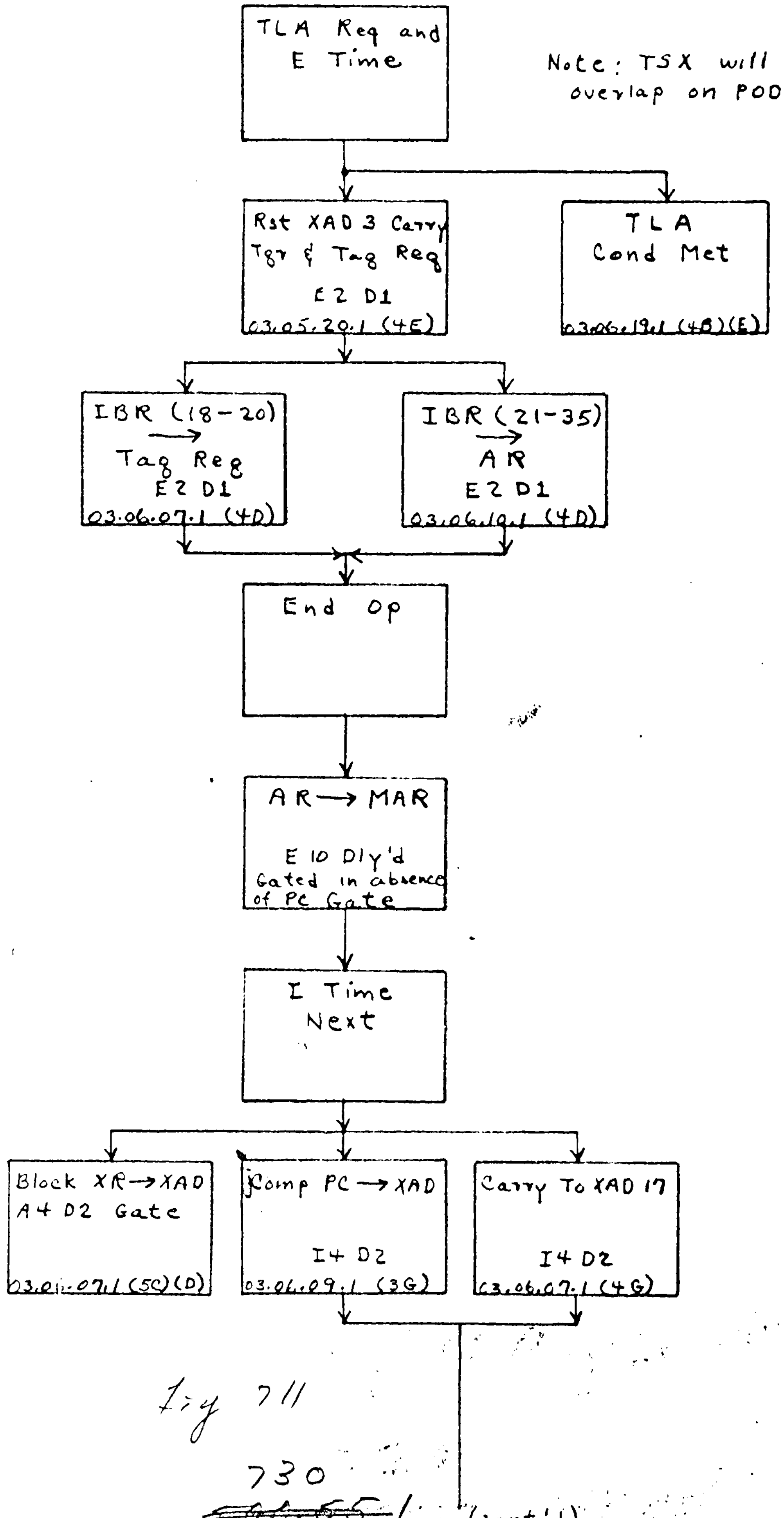
Pg 710 D

729

~~500, 51, 4~~

11-8-61 WEK

IBR TSX


TLA Req and E Time
Rst XAD 3 Carry Tgr & Tag Reg E2 D1 03.05.20.1 (4E)
TLA Cond Met 03.06.19.1 (4B)(E)
IBR (18-20) → Tag Reg E2 D1 03.06.07.1 (4D)
IBR (21-35) → AR E2 D1 03.06.10.1 (4D)
End Op
AR → MAR E 10 Dly'd Gated in absence of PC Gate
I Time Next
Block XR → XAD A4 D2 Gate 03.06.07.1 (5C)(D)
Comp PC → XAD I4 D2 03.06.09.1 (3G)
Carry To XAD 17 I4 D2 03.06.07.1 (4G)

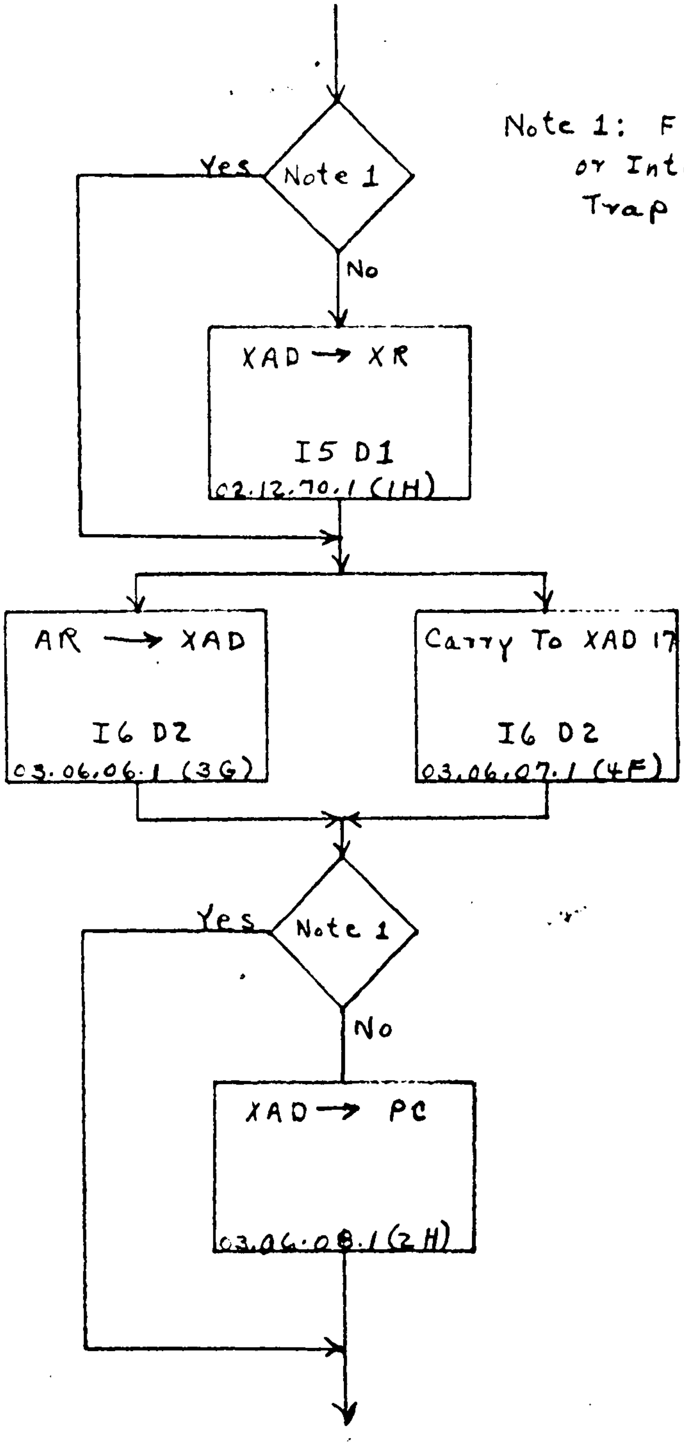
Note: TSX will not overlap on POD 34

fig 711

730
~~66,65~~ /

(cont'd)

11-8-61 WEK



Note 1: FP Trap or I-O Trap or Interrupt or Channel Trap Demand

Fig 7 11

731

~~800, 55~~, 2

IBR TRA, TTR

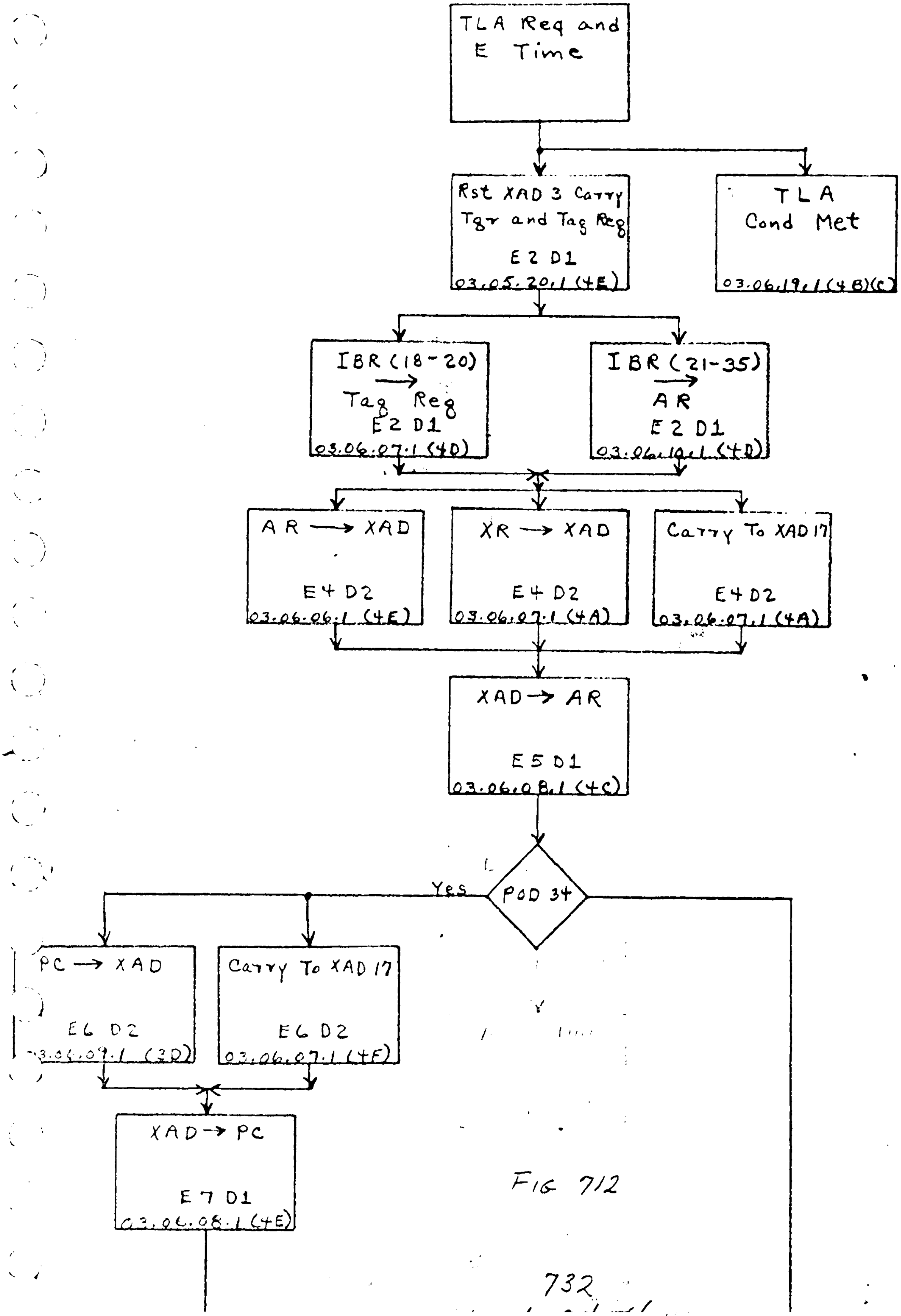


FIG 712

11-9-61 A (cont'd

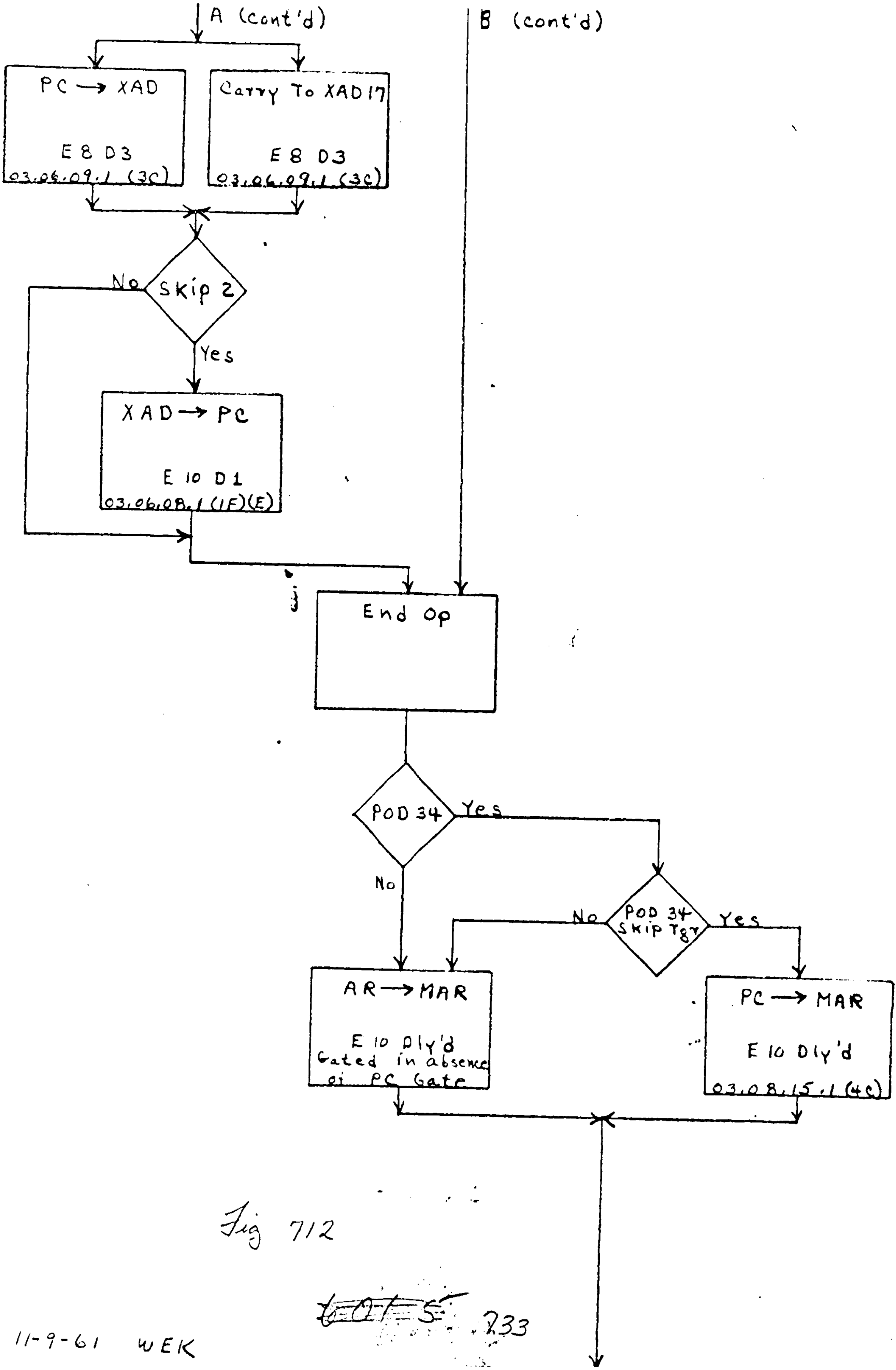
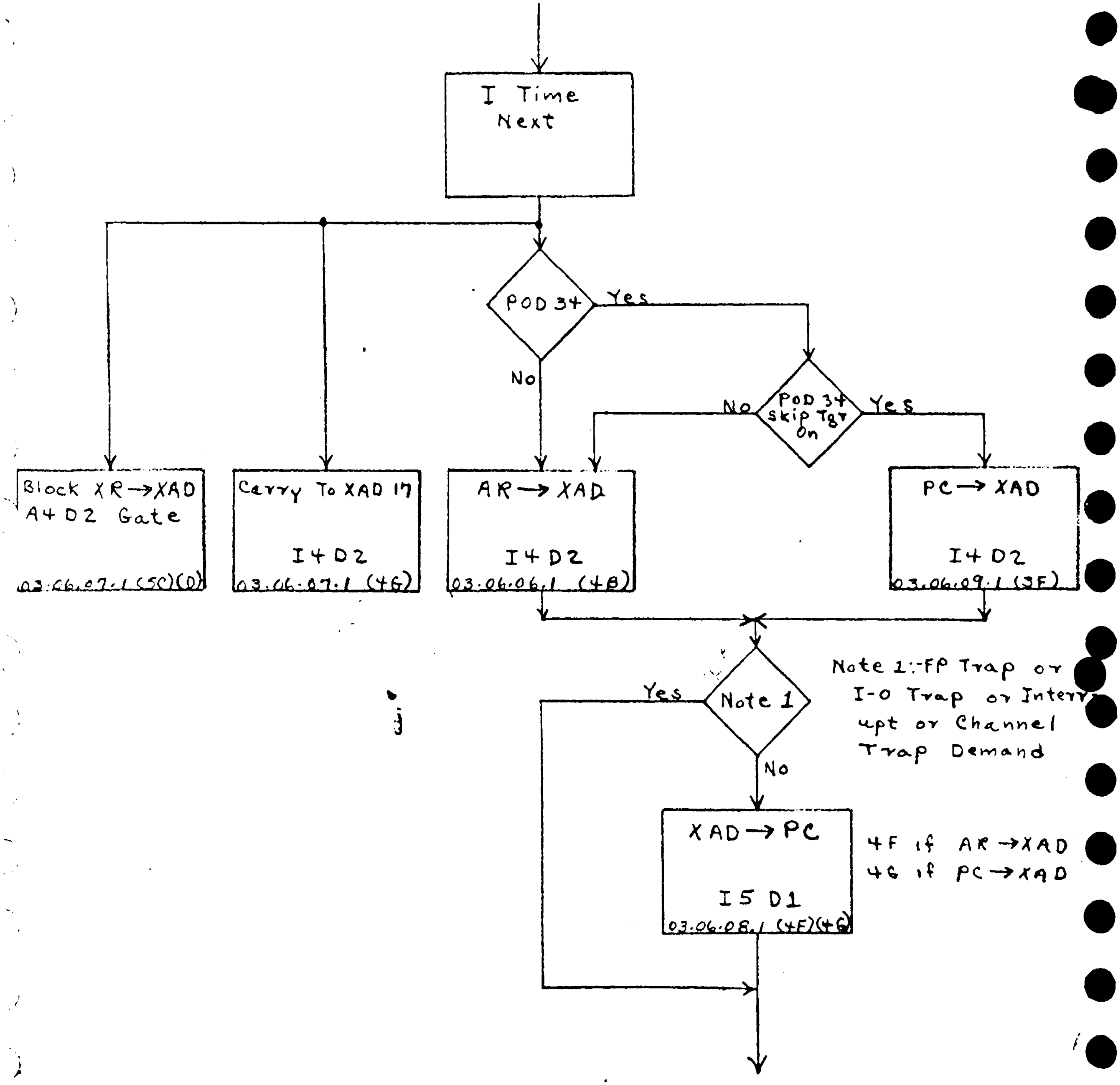


Fig 712

11-9-61 WEK

IBR TRA, TTR (cont'd)(2)



11-9-61 WEK

703. Store Lookahead

The timing rules for a store lookahead require that the instruction in the even location have an ~~xxx~~ octal code of 06xx (store type). When this is the case the execution time of the next cycle is reduced by one cycle. If the instruction following an 06xx octal code instruction is a one cycle instruction it is executed in overlapped operation and requires no cycle time. If the address of an 06xx instruction located at address 2n is 2n+1, no overlap is possible.

Without overlap the following program would require four cycles,

100 STO

101 CLA

The store instruction would require an I and an E cycle, as would the clear and add instruction.

```
_______I_______/_______E_______/_______I_______/_______E_______/
               STO             /              CLA
```

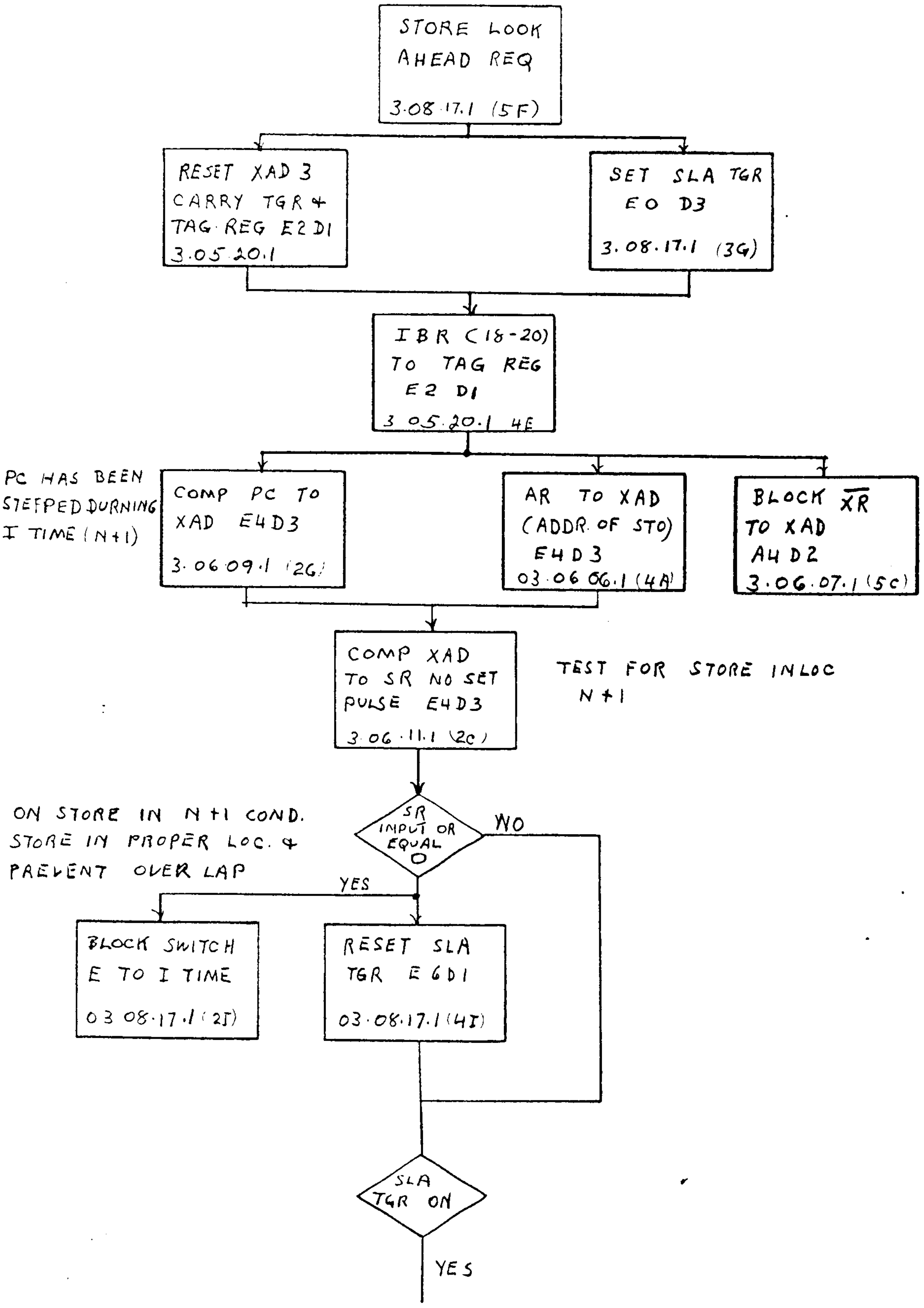
With overlap, the computer is forced to go to I time at the middle of the E cycle, operations are combined, and the two instructions completed in a total of three cycles.

```
_______I_______/______E-I______/_______E_______/
---------- STO---------
               -------------CLA--------------
```

The flow chart, Figure 713, shows the sequence of a store lookahead function. The first page shows that an SLA request must be initiated. The conditions necessary for an SLA request and for setting the SLA trigger are shown in Figure 706. The XAD 3 carry trigger and the tag register are reset and the contents of IBR 18-20 gated to the tag register at E2.

At E4(D3) the program counter and address register are compared to

STORE LOOK AHEAD



PC HAS BEEN STEPPED DURNING I TIME (N+1)

TEST FOR STORE IN LOC N+1

ON STORE IN N+1 COND. STORE IN PROPER LOC. + PREVENT OVER LAP

CONT'D PAGE 2

Fig 713

414

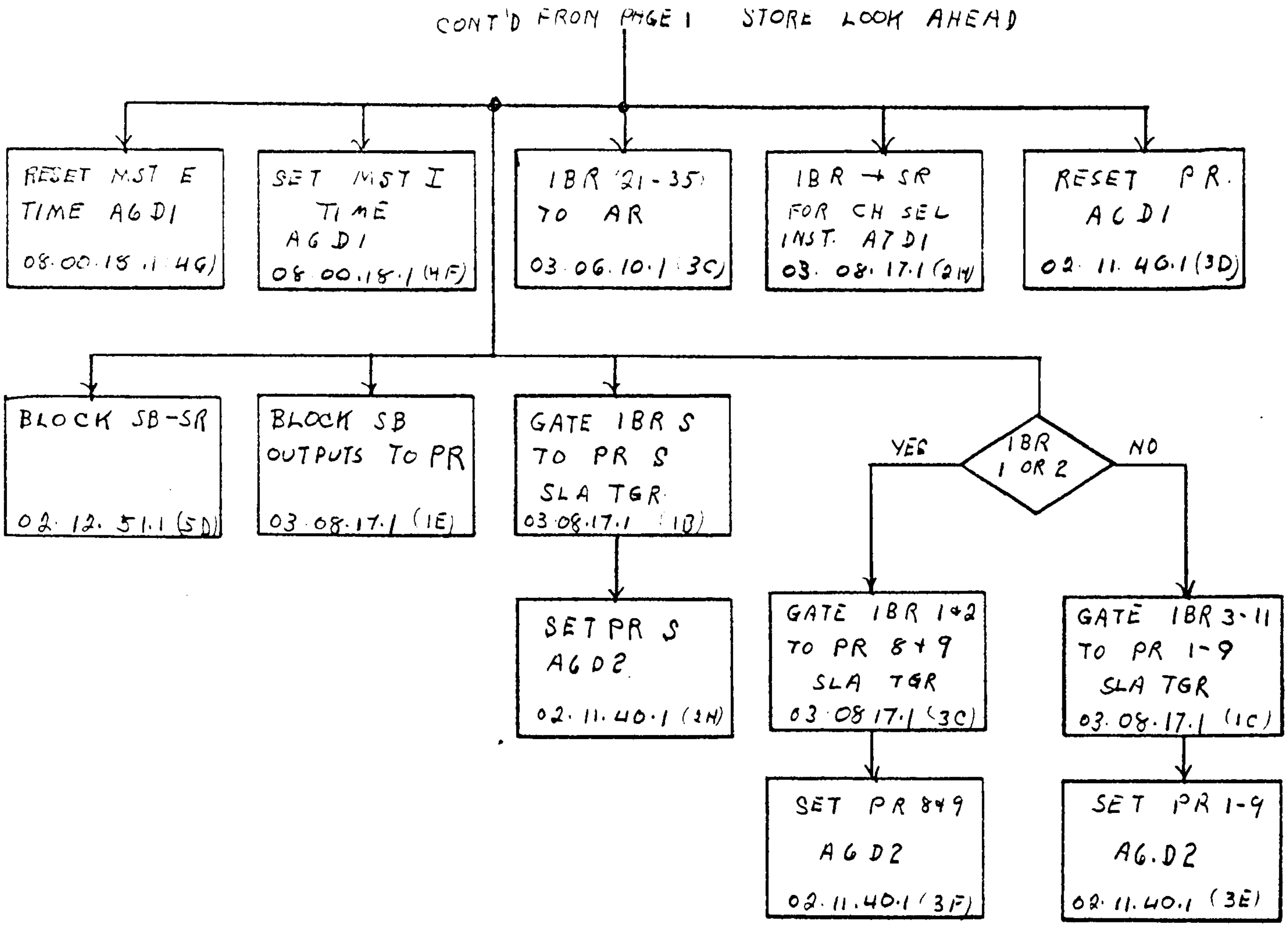

CONT'D FROM PAGE 1 STORE LOOK AHEAD
RESET MST E TIME A6 D1 08.00.18.1 (4G)
SET MST I TIME A6 D1 08.00.18.1 (4F)
IBR (21-35) TO AR 03.06.10.1 (3C)
IBR → SR FOR CH SEL INST. A7 D1 03.08.17.1 (2H)
RESET PR. A6 D1 02.11.40.1 (3D)
BLOCK SB-SR 02.12.51.1 (5D)
BLOCK SB OUTPUTS TO PR 03.08.17.1 (1E)
GATE IBR S TO PR S SLA TGR 03.08.17.1 (1B)
SET PR S A6 D2 02.11.40.1 (2H)
IBR 1 OR 2
YES
NO
GATE IBR 1+2 TO PR 8+9 SLA TGR 03.08.17.1 (3C)
GATE IBR 3-11 TO PR 1-9 SLA TGR 03.08.17.1 (1C)
SET PR 8+9 A6 D2 02.11.40.1 (3F)
SET PR 1-9 A6.D2 02.11.40.1 (3E)

Figure 713

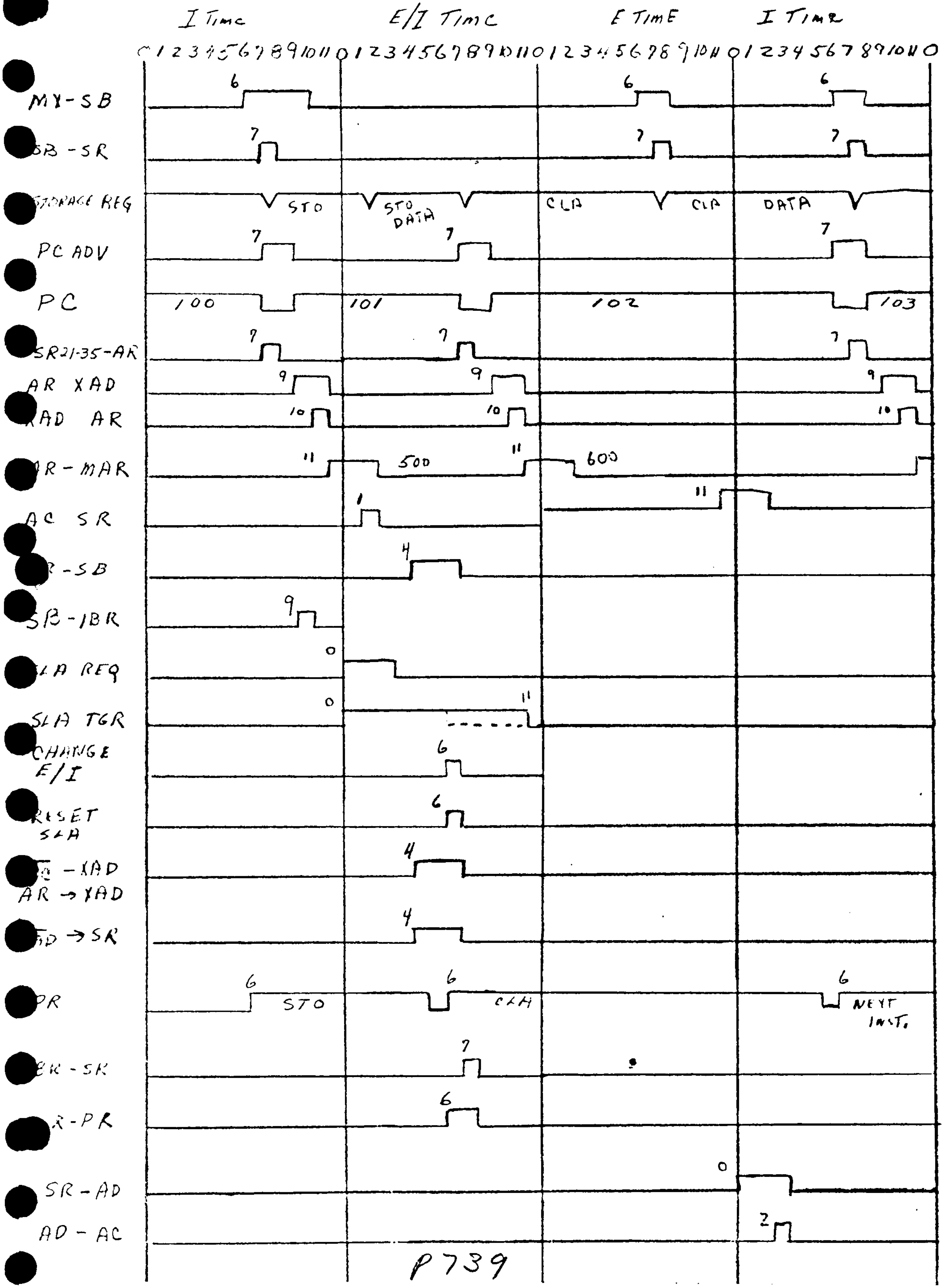
make certain the program is not salling for a store operation into the address of the next instruction. The result of the complement addition is gated to the first latch of the storage register and a zero check made. The information is not gated ~~in~~ into the storage register but sensed for zeros at the input OR.

If the input to the storage register contains all zeros, the overlap feature is nullified by blocking the switch to E time and resetting the SLA trigger. If the input is not zero, master E time is reset (Figure 311) and master I time set (Figure 306), putting the computer into I time.

The contents of the IBR are gated to the storage register and the program continues as though the computer were in a normal I cycle. Because the store operation is being carried out around this transition period (from E to I), the SB gating to the SR and the PR is blocked.

The CLA instruction will be completed during the following E cycle.

SLA Timing Chart



100 STO 5
101 CLA 6
102 NEXT INS

PC → MAR

NEXT ADDRES

NO SET
ZERO TEST

P739

MFS
5/22/6

6.0.00 IBM 7151 CONSOLE CONTROL UNIT

The 7151 Console Control Unit is a separate unit that may be placed at any convenient location within certain cable length restrictions. It provides manual and semiautomatic control over the system. The console consists of three panels: an operator's panel, a customer engineer's test panel, and a marginal check panel. With proper use, the console can be one of the customer engineer's most powerful tools. The time spent learning how to use it effectively is returned many times when diagnosing system errors.

This section introduces the keys, lamps, switches, and test facilities, their function, and any associated logic.

During the progress of a program, the operator may want some amount of control. For example, at a given point in a calculation, the computer is given the instruction to halt. The operator then can make a visual check of the information developed thus far in the program. At this point, one of several alternate manual steps may be performed, depending on the data observed. For this operation, the automatic-manual switch is set to MANUAL. With the machine in this state, the operator may enter and execute an instruction, interrogate any locations in storage for a visual check of the information stored, or load data from the operator's panel keys. After the desired manipulations have been made, the machine is returned to automatic status; the start key is depressed and the program continues.

The start and stop of the machine are under control of the master stop trigger. This trigger in turn controls "B cycle interrupt," which gates the I, E, and L cycles.

The keys, switches, and lamps on the console provide a means to:

1. Start or stop the machine
2. Step through a program at reduced speed
3. Check the status of the CPU
4. Display or revise the contents of storage
5. Alter the program

In addition, the customer engineer has facilities for several testing features which include:

1. Auxiliary start and reset key
2. I-O interlocks
3. Continuous execution of an instruction
4. Power jacks for test equipment
5. B cycle control

The console (Figure 6.0-1) is divided into three sections; an operator's panel, a customer engineer's test panel, and a bias panel. Figure 6.0-2 designates system page locations for the keys and indicators located on the console.

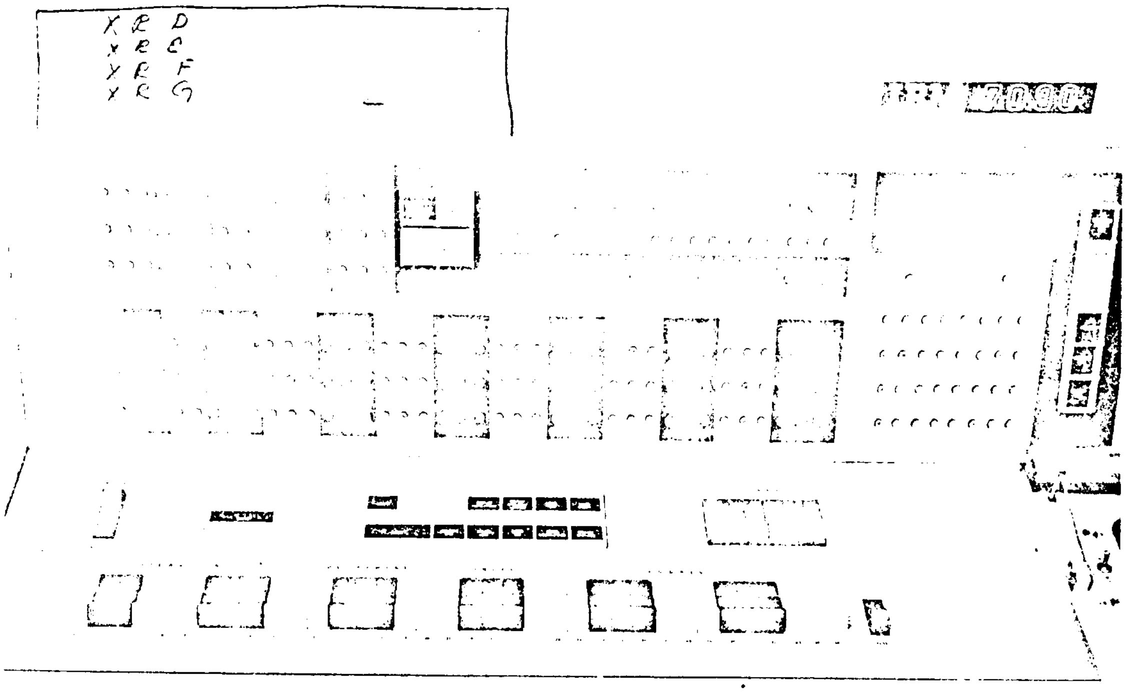


FIGURE 6.0-1. IBM 7151 CONSOLE

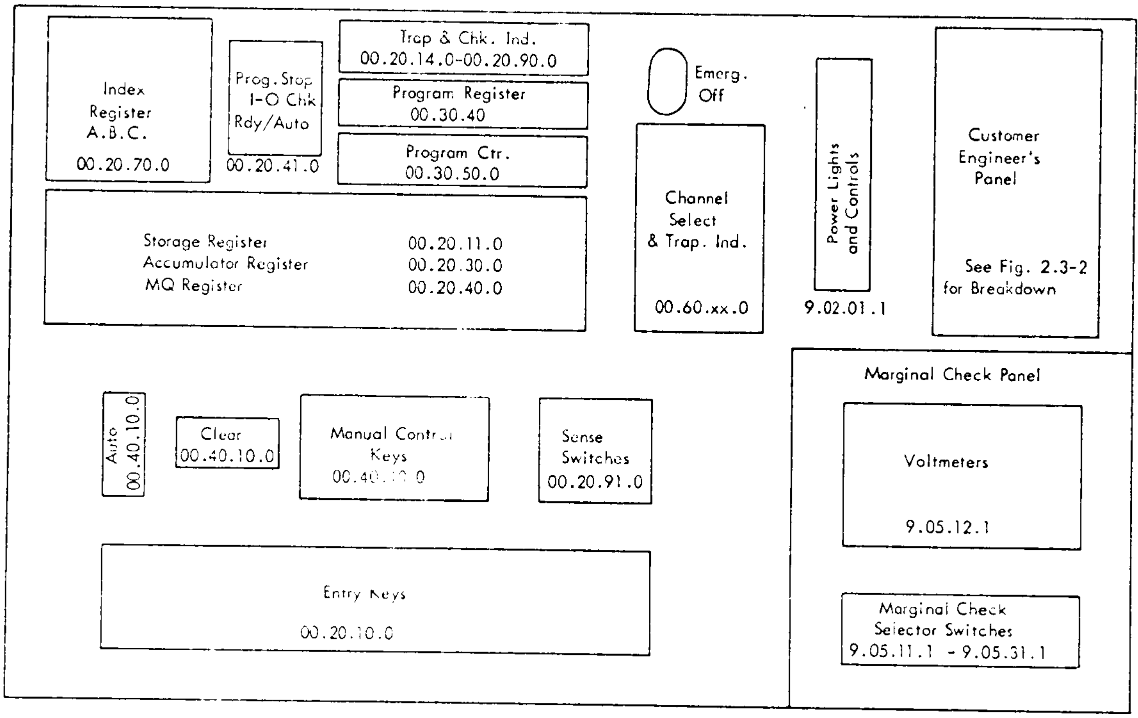


FIGURE 6.0-2. BLOCK DIAGRAM OF CONSOLE

~~6.1~~.00 OPERATOR'S PANEL

The operator's panel provides for visual checking of the information in the computer and for manual control of the computer's functions. It is also a station from which power may be applied to or removed from the system.

~~6.1~~.01 Indicators

All indicators on the console are of the incandescent type. When used to indicate the condition of a register, a lamp being on signifies a 1, while a lamp being off signifies a 0.

Internal Registers

The contents of the internal registers (accumulator, multiplexor-quotient, storage register, instruction counter, instruction register, and index registers A, B, and C) are displayed directly on the panel.

Trap

This lamp is on whenever the machine is in the transfer trapping mode.

Simulate

The simulate lamp is on when the 7090 is operating in any of the following modes associated with the 704, 709, or 7090 compatibility program:

1. I-O select and sense trap mode
2. Copy and locate drum address trap mode
3. Storage nullification mode

Accumulator Overflow

The accumulator overflow lamp is on any time during a fixed point operation (add, subtract, and so on) or shifting operation that a carry out of position 1 of the accumulator occurs. It is also turned on by a bit in position P during the execution of a floating point instruction while in compatibility mode. It may be turned off by the TNO or TOV instruction.

Quotient Overflow

This lamp is on whenever the computer is using the compatibility program and an MQ overflow occurs.

Read-Write Select

The read-write (R-W) select lamp is on whenever the channel-in-use trigger is on in any data channel.

Divide Check

The divide check lamp is turned on in fixed point division if the dividend accumulator (AC) is greater than or equal to the divisor (storage register). In floating-point divide the lamp is on if the magnitude of the fraction of the dividend is greater than or equal to twice the magnitude of the fraction of the divisor. The indicator may be tested by the DCT instruction.

Channel Select (A-H)

The channel select (A-H) lamps, one for each channel, are on according to the data channel that is in operation. They are off when the channel is not in operation.

Command Word Trap (Channels A-H)

These lamps are turned on according to the corresponding channel that is enabled to trap on command word or end of file. The lamp for a particular channel is turned off when the channel is disabled.

Tape Check Trap (Channels A-H)

These lamps are on according to the corresponding channel that is enabled to trap on a tape check. The lamp for a particular channel is off when the channel is disabled.

Channel Tape Check

These lamps, one for each channel, are on if any error is detected while writing or reading. The lights may be turned off by the execution of a Transfer on Redundancy Check instruction.

Trap Control

This lamp is on when the channel is not executing a channel trap. It is off when any channel enters a trap condition. While the light is off, no channel traps may be executed. Channel traps may be executed only when the light is on at the same time as any of the enabled lamps.

Program Stop (Red)

The program stop lamp is turned on whenever the computer executes a halt instruction and no data channels are in operation (DVH or VDH excepted).

I-O Check (White)

The I-O check lamp may be turned on by any of the following conditions:

1. If a RCH or LCH is decoded and the specified data channel has not been selected.
2. If, when writing, a data channel data register has not been loaded with a word from storage by the time its contents are to be sent to the output unit.

3. If, when reading, a data channel data register has not transmitted its contents to storage by the time new information is to be loaded into it from an output unit.

The I-O check lamp may be turned off by the execution of an IOT.

Ready Light (White)

The ready lamp comes on after power comes up and remains on except when the machine is in automatic status, the continuous enter instruction switch is on, the I-O interlock switch is in manual, or if any B-time control switch is on.

Automatic (Yellow)

The automatic lamp is on whenever the computer is executing instructions in the automatic mode or whenever a data channel is in operation.

Console Power-On (Red)

The console power-on lamp is on whenever power is applied to the console.

Central Components Power Check (Yellow)

The central components power check comes on whenever a fuse or circuit breaker opens in CPU frame 1 or 2, multiplexor, or core storage. It also lights when core storage has improper oil temperature or low oil pressure.

I-O Power Check (Yellow)

The I-O power check lamp is turned on whenever a fuse, an open thermal, or airflow failure is sensed in a data channel.

Power (Red)

The power indicator is turned on whenever DC power is up in core storage.

Marginal Check +6 (Yellow)

This indicator is on whenever the +6 supply marginal check variable autotransformer is not in the home position.

Marginal Check -12 (Yellow)

This indicator is on whenever the -12 supply marginal check variable autotransformer is not in the home position.

~~6.1.~~02 Manual Controls

Figure 6.1-1 shows the keys and switches on the operator's panel that provide for starting and stopping the machine and initiating computer functions. All keys are of the spring-returned variety with the exception of the auto-manual key, entry keys, sense switches, and emergency-off key.

Power-On

With the system in normal-off status, pressing the power-on key will commence the power-on sequence. Ready status (power-on) will be reached in about 20 seconds. As power comes on, a clear operation is performed, resetting all registers and triggers, and setting memory to all zeros.

Normal-Off

The normal-off key will initiate the following:

1. Immediate removal of 60-cycle power from the MG set, MG blower, and all frame blowers except memory.
2. Immediate removal of 400-cycle power from the 30-60 volt memory power supply.
3. After 5 seconds, removal of 400-cycle power from the standard memory supply.
4. After 3 minutes, removal of power from the memory blowers.

The dropping of the 60-cycle power removes the input to the MG, but the MG still rotates at about full speed for longer than 5 seconds, allowing the memory power to sequence down.

Emergency-Off

When the emergency-off switch is pulled, all power is immediately removed from the 709~~0~~ system, except the voltage to HR 24 and HR 30 points in the power control unit. The emergency-off switch is to be used only in emergencies, because of possible damage to circuits.

Reset~~s~~ ✓

The reset circuits control the resetting of various components in the system. There are three types of resets which may be initiated from the panel.

> An interlock reset is initiated by the load keys, clear key, and power-on key. It causes the resetting of all registers (except sense indicator), panel light (except power on and ready), all channels that are in operation and their associated registers.
>
> An operator's reset is initiated by the reset key. It performs all the functions of an interlock reset plus a bias reset for CPU triggers.
>
> A power-on reset occurs when power is applied to the system. This reset accomplishes an interlock reset, an operator's reset, plus the resetting of the clock and setting core storage to all zeros. The clear key also initiates a power-on reset if the machine is in automatic status.

Automatic-Manual Key

The auto-manual switch controls the rest of the switches on the console. If a program is running in automatic status, and the switch is put in the manual position, the program will stop after it completes the operation it is performing. If an I-O program is running, I-O will complete the operation before stopping. The auto-manual switch will not affect the program stop status.

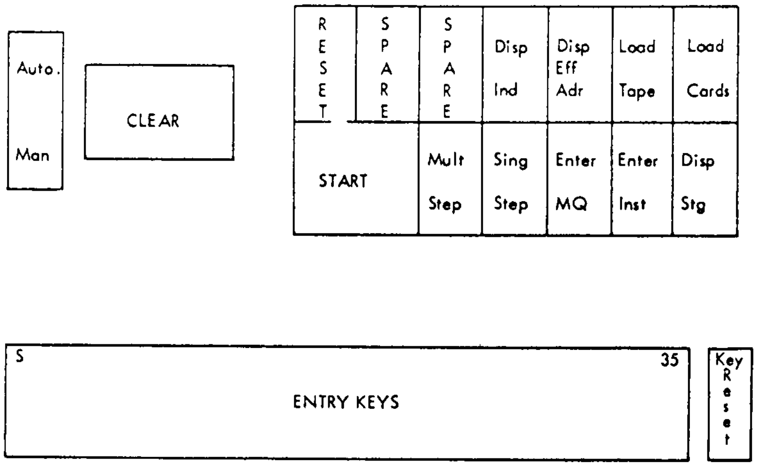


FIGURE 6.1-1. OPERATOR'S PANEL

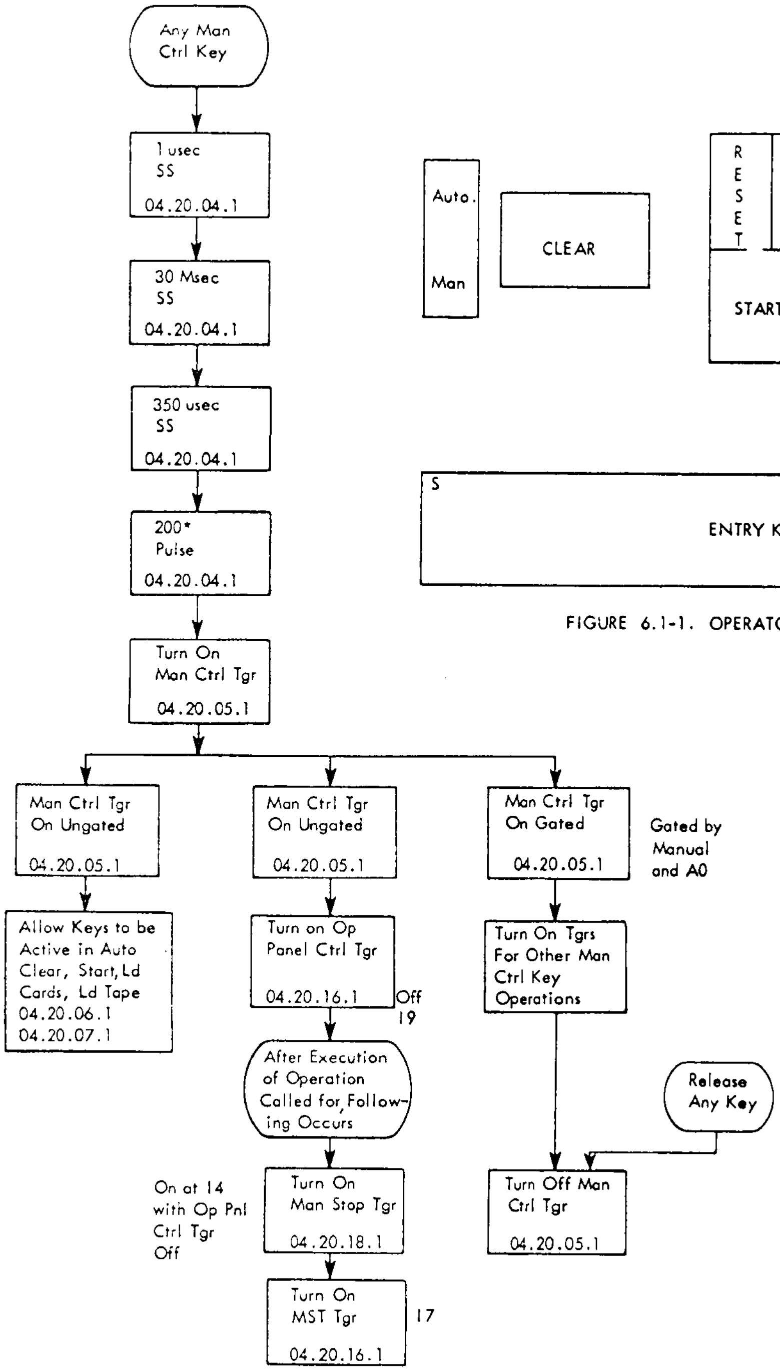

Any Man Ctrl Key
1 usec SS 04.20.04.1
30 Msec SS 04.20.04.1
350 usec SS 04.20.04.1
200* Pulse 04.20.04.1
Turn On Man Ctrl Tgr 04.20.05.1
Man Ctrl Tgr On Ungated 04.20.05.1
Man Ctrl Tgr On Ungated 04.20.05.1
Man Ctrl Tgr On Gated 04.20.05.1
Gated by Manual and A0
Allow Keys to be Active in Auto Clear, Start, Ld Cards, Ld Tape 04.20.06.1 04.20.07.1
Turn on Op Panel Ctrl Tgr 04.20.16.1
Off 19
Turn On Tgrs For Other Man Ctrl Key Operations
After Execution of Operation Called for, Following Occurs
Release Any Key
On at 14 with Op Pnl Ctrl Tgr Off
Turn On Man Stop Tgr 04.20.18.1
Turn Off Man Ctrl Tgr 04.20.05.1
Turn On MST Tgr 04.20.16.1
17

FIGURE 6.1-2. MANUAL CONTROL KEY

Entry Keys

There are 36 entry keys on the operator's panel: S, 1-35. Depressing a key sets a 1 in that position; leaving a key normal sets a 0 in that position. Information set in the entry keys may be entered into storage, executed, or used for a storage inquiry address. Also into MQ with the enter MQ key. The entry keys may all be reset to 0 by depressing the reset key to the right of position 35. The contents of these keys will be set into the SR, instruction register and tag register when using the ENK or when using the continuous enter instruction switch and start, single-step, machine cycle or multiple step keys.

~~6.1.03~~ Manual Control Keys

When any manual control key is pressed, a series of single-shots and triggers are set. Separating the key from the usable signal prevents false indications from noise generated by the key (Figure 6.1-2). Three single-shots are fired in sequence: a 1-usec, a 20-ms and a 350-usec. The 350-usec single-shot is taken through a delay network to develop a 200-ns pulse that turns on the manual control trigger. By this time, the switch has settled down and the trigger to perform the desired operation turns on, resetting the manual control trigger.

Start Key (Figure 6.1-3)

If the CPU is in automatic and ready status, pressing the start key initiates machine operation by turning off the master stop trigger and the program stop trigger. The master stop trigger off conditions "not B cycle interrupt," allowing the computer to proceed. If the system is in manual status, pressing the start key turns off the program stop trigger.

Clear Key (Figure 6.1-4)

The clear key is only operative if the computer is in automatic status. Pressing the clear key: (performs all functions of reset key in addition to below)

1. Fires a 1-usec single-shot to reset the clock and all channel registers of channels that are not in manual status.
2. Resets CPU interlocks and registers.
3. Conditions circuits which allow zeros to be written into all storage locations.
4. Resets SI register. 5. Turns on multiple tag mode indicator.

The program counter controls the stepping through memory, with position 2 of the PC indicating when all addresses have been zeroed. The turn-on of PC2 trigger will turn on the master stop trigger at E10.

Display Storage (Figure 6.1-5)

Pressing the display storage key causes the address portion of the operator's panel keys to be sensed, to determine the address in storage to be displayed. All 36-bit positions of the address will be displayed in the storage register. This is accomplished by: (1) turning off the master stop trigger, (2) bringing the operator's panel keys to the storage register in I time, and (3) suppressing "storage bus to storage register." The address portion of the SR is routed through the adders and address switches, to the address register. During E time, the SB is gated to the SR, which contains the 36 bits of the desired address. If a tag or indirect addressing is specified in the operator's keys, the contents of the effective address or the I-A address will be displayed. Effective in manual status only

Display Indicators

The display indicators key gates the true value of the sense indicators to the storage for display (Figure 6.1-6). Information remains displayed until another operation involving SR is performed or reset key ~~dep~~ depressed.

register indicators

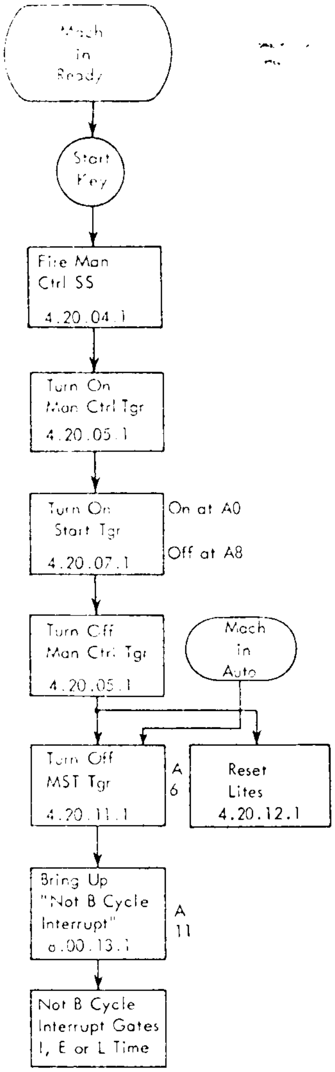

Mach in Ready
Start Key
Fire Man Ctrl SS 4.20.04.1
Turn On Man Ctrl Tgr 4.20.05.1
Turn On Start Tgr 4.20.07.1
On at A0
Off at A8
Turn Off Man Ctrl Tgr 4.20.05.1
Mach in Auto
Turn Off MST Tgr 4.20.11.1
A 6
Reset Lites 4.20.12.1
Bring Up "Not B Cycle Interrupt" 8.00.13.1
A 11
Not B Cycle Interrupt Gates I, E or L Time

FIGURE 6.1-3. START

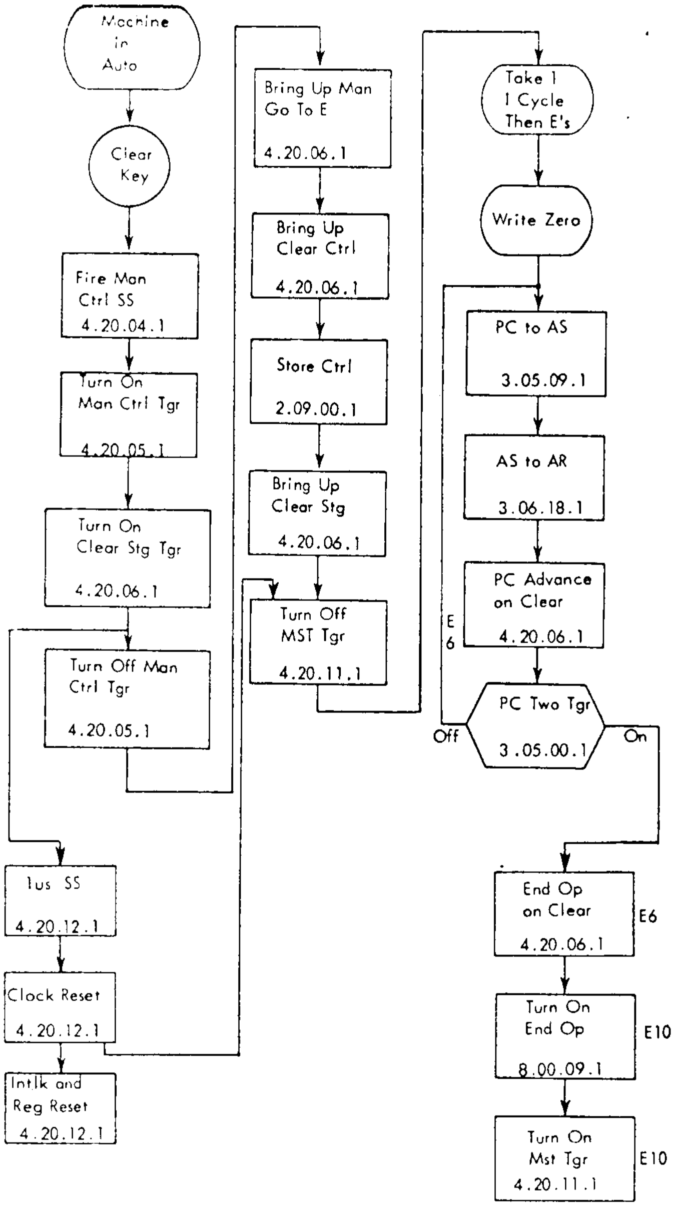

Machine in Auto
Clear Key
Fire Man Ctrl SS 4.20.04.1
Turn On Man Ctrl Tgr 4.20.05.1
Turn On Clear Stg Tgr 4.20.06.1
Turn Off Man Ctrl Tgr 4.20.05.1
Ius SS 4.20.12.1
Clock Reset 4.20.12.1
Intlk and Reg Reset 4.20.12.1
Bring Up Man Go To E 4.20.06.1
Bring Up Clear Ctrl 4.20.06.1
Store Ctrl 2.09.00.1
Bring Up Clear Stg 4.20.06.1
Turn Off MST Tgr 4.20.11.1
Take 1 I Cycle Then E's
Write Zero
PC to AS 3.05.09.1
AS to AR 3.06.18.1
PC Advance on Clear 4.20.06.1
E 6
PC Two Tgr 3.05.00.1
Off
On
End Op on Clear 4.20.06.1
E6
Turn On End Op 8.00.09.1
E10
Turn On Mst Tgr 4.20.11.1
E10

FIGURE 6.1-4. CLEAR STORAGE

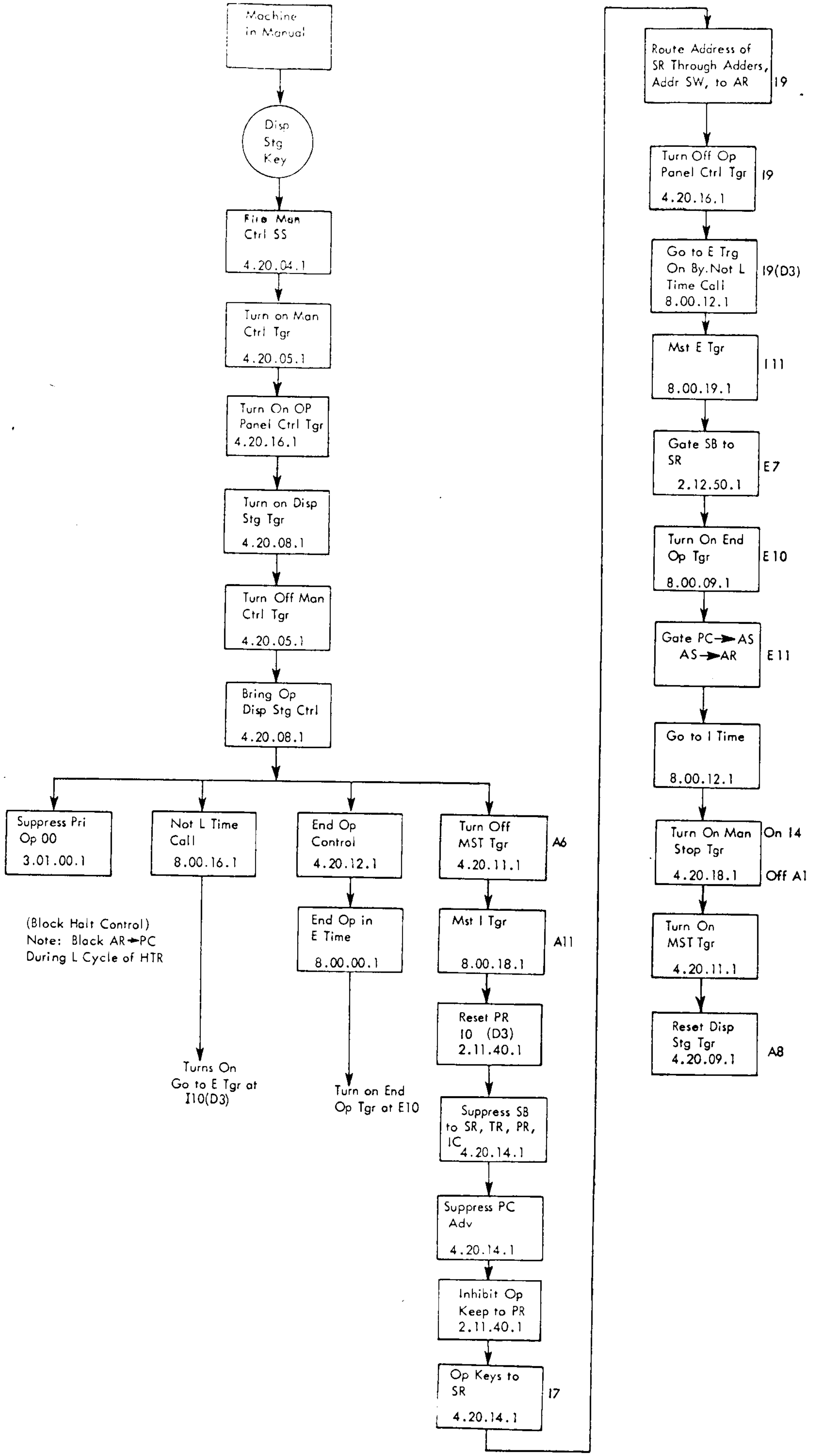
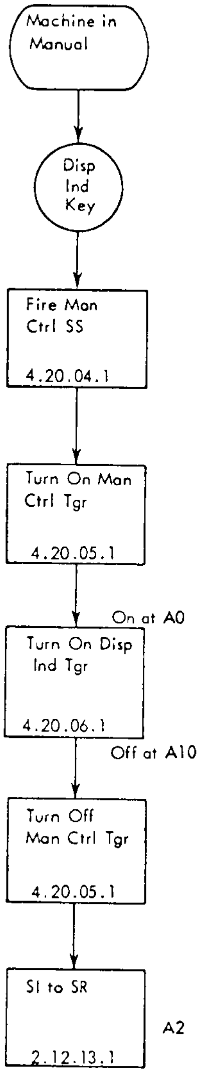
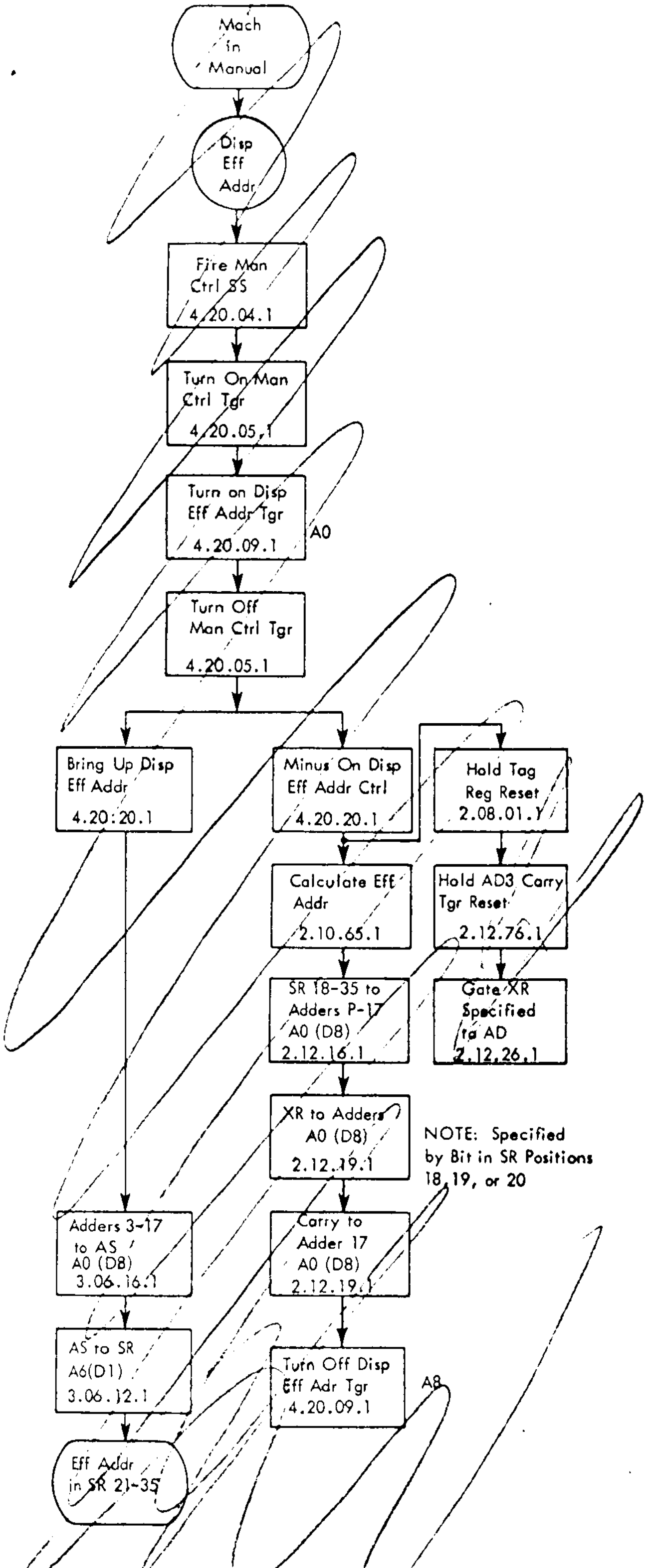

Machine in Manual
Disp Stg Key
Fire Man Ctrl SS 4.20.04.1
Turn on Man Ctrl Tgr 4.20.05.1
Turn On OP Panel Ctrl Tgr 4.20.16.1
Turn on Disp Stg Tgr 4.20.08.1
Turn Off Man Ctrl Tgr 4.20.05.1
Bring Op Disp Stg Ctrl 4.20.08.1
Suppress Pri Op 00 3.01.00.1
(Block Halt Control)
Note: Block AR→PC During L Cycle of HTR
Not L Time Call 8.00.16.1
Turns On Go to E Tgr at I10(D3)
End Op Control 4.20.12.1
End Op in E Time 8.00.00.1
Turn on End Op Tgr at E10
Turn Off MST Tgr 4.20.11.1
A6
Mst I Tgr 8.00.18.1
A11
Reset PR 10 (D3) 2.11.40.1
Suppress SB to SR, TR, PR, IC 4.20.14.1
Suppress PC Adv 4.20.14.1
Inhibit Op Keep to PR 2.11.40.1
Op Keys to SR 4.20.14.1
I7
Route Address of SR Through Adders, Addr SW, to AR
I9
Turn Off Op Panel Ctrl Tgr 4.20.16.1
I9
Go to E Trg On By Not L Time Call 8.00.12.1
I9(D3)
Mst E Tgr 8.00.19.1
I11
Gate SB to SR 2.12.50.1
E7
Turn On End Op Tgr 8.00.09.1
E10
Gate PC→AS AS→AR
E11
Go to I Time 8.00.12.1
Turn On Man Stop Tgr 4.20.18.1
On I4
Off A1
Turn On MST Tgr 4.20.11.1
Reset Disp Stg Tgr 4.20.09.1
A8

FIGURE 6.1-5. DISPLAY STORAGE



Machine in Manual
Disp Ind Key
Fire Man Ctrl SS 4.20.04.1
Turn On Man Ctrl Tgr 4.20.05.1
On at A0
Turn On Disp Ind Tgr 4.20.06.1
Off at A10
Turn Off Man Ctrl Tgr 4.20.05.1
SI to SR 2.12.13.1
A2

FIGURE 6.1-6. DISPLAY INDICATORS

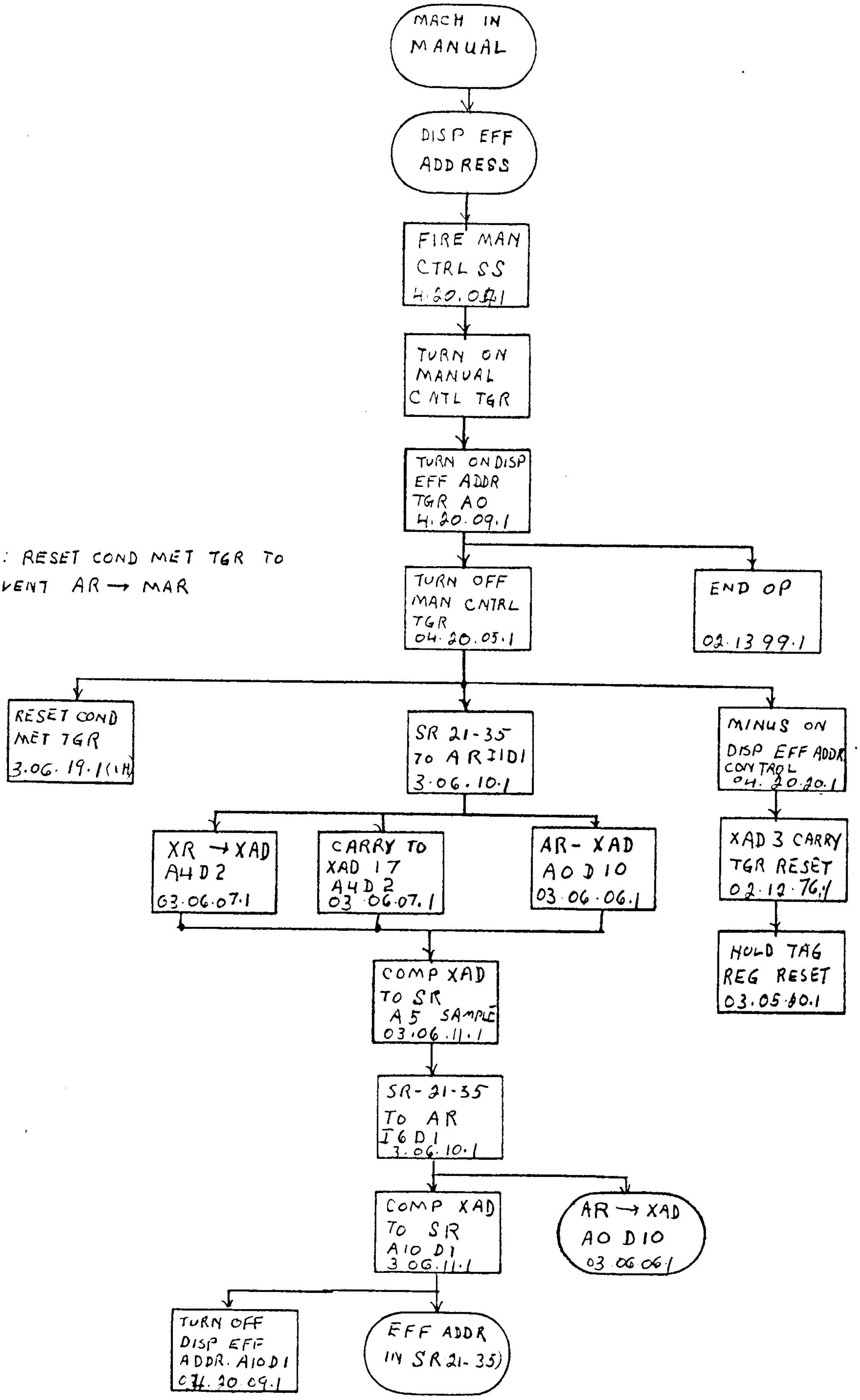


Mach in Manual
Disp Eff Addr
Fire Man Ctrl SS 4.20.04.1
Turn On Man Ctrl Tgr 4.20.05.1
Turn on Disp Eff Addr Tgr 4.20.09.1
A0
Turn Off Man Ctrl Tgr 4.20.05.1
Bring Up Disp Eff Addr 4.20.20.1
Minus On Disp Eff Addr Ctrl 4.20.20.1
Hold Tag Reg Reset 2.08.01.1
Calculate Eff Addr 2.10.65.1
Hold AD3 Carry Tgr Reset 2.12.76.1
SR 18-35 to Adders P-17 A0 (D8) 2.12.16.1
Gate XR Specified to AD 2.12.26.1
XR to Adders A0 (D8) 2.12.19.1
NOTE: Specified by Bit in SR Positions 18, 19, or 20
Adders 3-17 to AS A0 (D8) 3.06.16.1
Carry to Adder 17 A0 (D8) 2.12.19.1
AS to SR A6(D1) 3.06.12.1
Turn Off Disp Eff Adr Tgr 4.20.09.1
A8
Eff Addr in SR 21-35

FIGURE 6.1-7. DISPLAY EFFECTIVE ADDRESS

DISPLAY EFFECTIVE ADDRESS

NOTE: RESET COND MET TGR TO PREVENT AR → MAR


MACH IN MANUAL
DISP EFF ADDRESS
FIRE MAN CTRL SS 4.20.01
TURN ON MANUAL CNTL TGR
TURN ON DISP EFF ADDR TGR A0 4.20.09.1
TURN OFF MAN CNTRL TGR 04.20.05.1
END OP 02.13 99.1
RESET COND MET TGR 3.06.19.1(1H)
SR 21-35 TO AR I1D1 3.06.10.1
MINUS ON DISP EFF ADDR CONTROL 04.20.20.1
XR → XAD A4D2 03.06.07.1
CARRY TO XAD 17 A4D2 03 06.07.1
AR - XAD A0 D10 03.06.06.1
XAD 3 CARRY TGR RESET 02.12.76.1
HOLD TAG REG RESET 03.05.60.1
COMP XAD TO SR A5 SAMPLE 03.06.11.1
SR-21-35 TO AR I6D1 3.06.10.1
COMP XAD TO SR A10 D1 3.06.11.1
AR → XAD A0 D10 03.06.06.1
TURN OFF DISP EFF ADDR. A10D1 04.20.09.1
EFF ADDR IN SR21-35)

EAS 7/9/62

Display Effective Address (~~Figure~~ 6.1-7) see new flow chart

The display effective address key initiates the calculation of the effective address of the instruction in the storage register. The actual address from the storage register is combined with the two's complement of the specified index register to produce the effective address. The calculated address is then placed in the address portion of the storage register. Positions 51 through 20 of the storage register are set to zero. The effective address may be calculated only once because the positions 18, 19, and 20 of the storage register have been set to zero, an indication of no index register. Manual only

Single Step (Figure 6.1-8)

Pressing the single-step key results in executing the instruction whose address is in the instruction counter prior to key depression. The instruction counter will be advanced, or altered, under control of the instruction executed, once for each time the key is pressed. If an I-O operation is executed, the ~~machine will continue to execute instruc-~~ tions at high speed until the end of the I-O operation. If the continuous enter instruction switch is on, the single-step key is pressed, and the 7090 is in manual status, the instruction set in the OP keys will be executed once. The single-step key is effective only if the system is in manual status, and not in program stop status or automatic.

the end 10 instructions -tch

Multiple Step (Figure 6.1-9)

This key is effective only in manual status with the program stop trigger on. The multiple-step key causes the repetition of single-step operations. The rate of operation is under control of a toggle switch located on the customer engineer test panel. The operator has the choice of low speed with a delay of 104 milliseconds between each instruction, or high speed with a delay of 24 milliseconds between each instruction. The program will continue to run as long as the multiple step key is pressed, or until a program halt is decoded.

50 ± 10 instructions at high speed, 10 ± 2 instructions at low speed,

Enter MQ Key

The enter MQ key provides a means for loading the MQ register from the operator's entry keys. The information may then be loaded into storage by placing the instruction STQ along with the desired address in the entry keys and depressing the enter instruction key. The enter MQ key is effective only when in manual status. See Figure 6.1-10.

Enter Instruction (Figure 6.1-11)

With the machine in manual status, the enter instruction key executes completely and correctly any legitimate instruction entered in the operator's panel entry keys. The contents of the instruction counter remain unchanged when an instruction is executed (except for a transfer or a skip type of instruction). The key is effective only in manual status. or trap

Load Cards (Figure 6.1-12)

~~The load cards key causes a reset of the instruction counter, address register, program stop trigger, simulate, and all channels not in manual status. Pressing the load cards key then causes a select of the card reader on channel A, reads the first three words, and proceeds to storage locations zero for the next command word. This is accomplished by bringing up "load ctrl," which fires a 1-usec single-shot. The auto load trigger goes on in the CPU, selecting the card reader on channel A. The word counter is set to three, and indicator S is turned on in channel. Three words from the first~~

see p 811A

LOAD CARDS AND LOAD TAPE

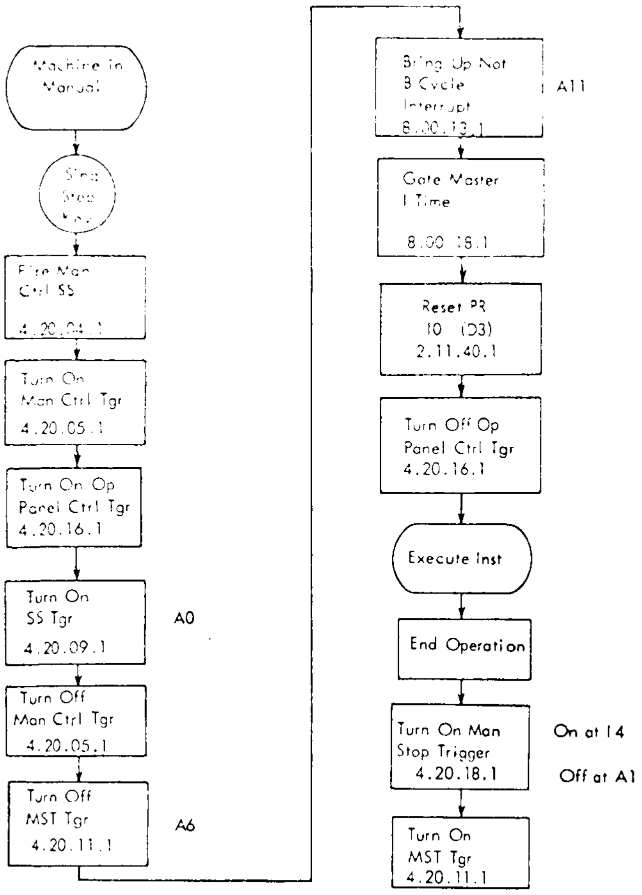
Depression of one of these keys results in storing the first three words from either the card reader or tape unit number one on channel A into storage addresses 0, 1 and 2, providing data channel A with the first word as an I/O command, and starting the computer with the second word stored as its first instruction. The computer must be in automatic status and the ready light on for peoper performance. Depression of a load key will then :

1. Reset the instruction counter, address register, program stop light, simulate light and all indicators and registers in all channels in automatic status.

2. Set card reader select (or tape ~~select~~ read select and unit select 1), control indicator S and word counter indicators 16 and 17, in channel A (if channel A is attached and in automatic status).

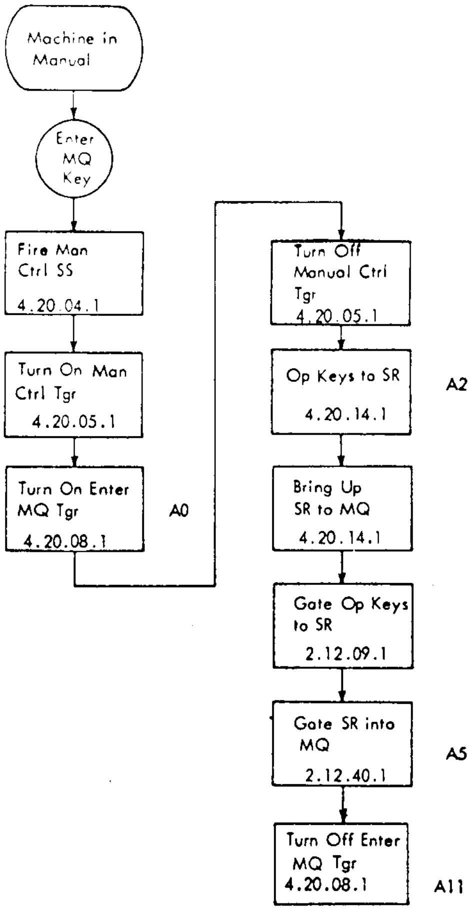
3. Channel A will normally store three words and then read a command from storage address 000000,

4. As channel A reads out its command, the master stop trigger in the computer should go to off and address register position 17 should be set on, thus starting the computer with the instruction at address 00001.



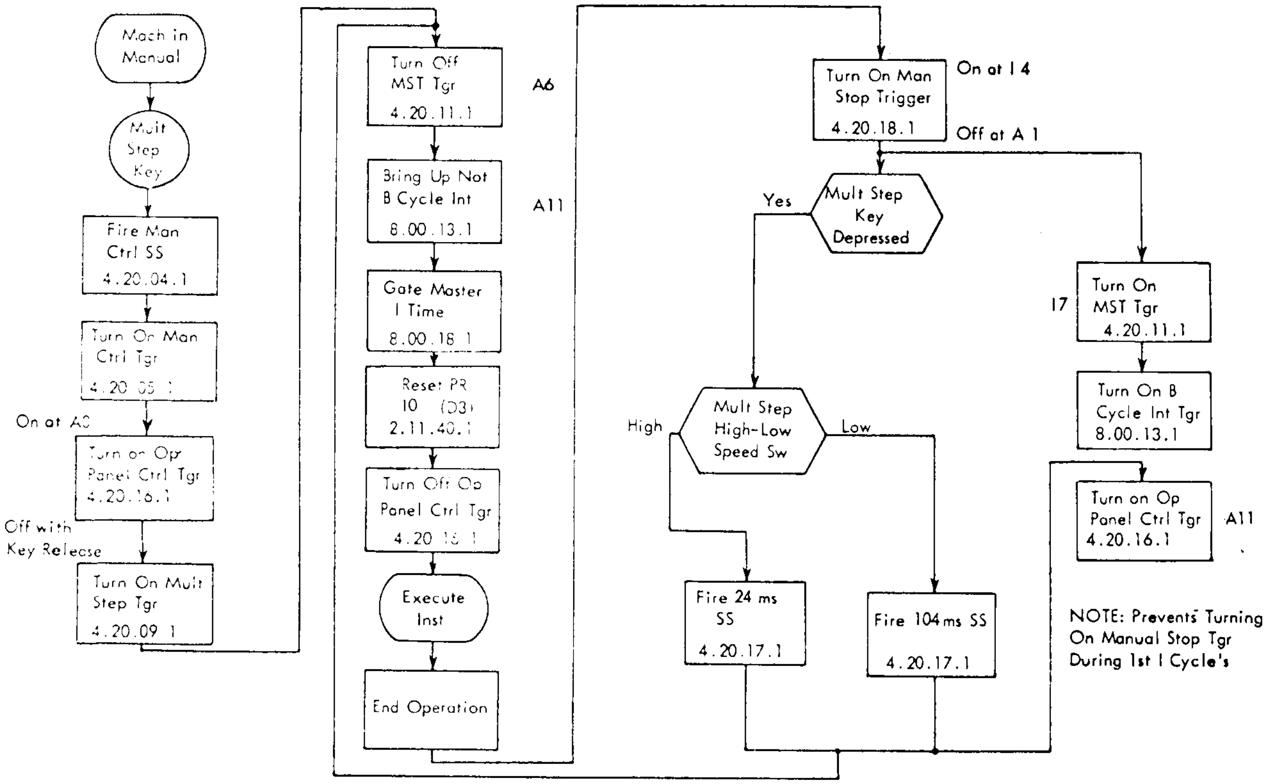
Machine in Manual
Sing Step Key
Fire Man Ctrl SS 4.20.04.1
Turn On Man Ctrl Tgr 4.20.05.1
Turn On Op Panel Ctrl Tgr 4.20.16.1
Turn On SS Tgr 4.20.09.1
A0
Turn Off Man Ctrl Tgr 4.20.05.1
Turn Off MST Tgr 4.20.11.1
A6
Bring Up Not B Cycle Interrupt 8.00.13.1
A11
Gate Master I Time 8.00.18.1
Reset PR 10 (D3) 2.11.40.1
Turn Off Op Panel Ctrl Tgr 4.20.16.1
Execute Inst
End Operation
Turn On Man Stop Trigger 4.20.18.1
On at I 4
Off at A1
Turn On MST Tgr 4.20.11.1

FIGURE 6.1-8. SINGLE STEP KEY



Machine in Manual
Enter MQ Key
Fire Man Ctrl SS 4.20.04.1
Turn On Man Ctrl Tgr 4.20.05.1
Turn On Enter MQ Tgr 4.20.08.1
A0
Turn Off Manual Ctrl Tgr 4.20.05.1
Op Keys to SR 4.20.14.1
A2
Bring Up SR to MQ 4.20.14.1
Gate Op Keys to SR 2.12.09.1
Gate SR into MQ 2.12.40.1
A5
Turn Off Enter MQ Tgr 4.20.08.1
A11

FIGURE 6.1-10. ENTER MQ



Mach in Manual
Mult Step Key
Fire Man Ctrl SS 4.20.04.1
Turn On Man Ctrl Tgr 4.20.05.1
On at A0
Turn on Op Panel Ctrl Tgr 4.20.16.1
Off with Key Release
Turn On Mult Step Tgr 4.20.09.1
Turn Off MST Tgr 4.20.11.1
A6
Bring Up Not B Cycle Int 8.00.13.1
A11
Gate Master I Time 8.00.18.1
Reset PR 10 (D3) 2.11.40.1
Turn Off Op Panel Ctrl Tgr 4.20.16.1
Execute Inst
End Operation
Turn On Man Stop Trigger 4.20.18.1
On at I 4
Off at A 1
Yes
Mult Step Key Depressed
Mult Step High-Low Speed Sw
High
Low
Fire 24 ms SS 4.20.17.1
Fire 104ms SS 4.20.17.1
17
Turn On MST Tgr 4.20.11.1
Turn On B Cycle Int Tgr 8.00.13.1
Turn on Op Panel Ctrl Tgr 4.20.16.1
A11
NOTE: Prevents Turning On Manual Stop Tgr During 1st I Cycle's

FIGURE 6.1-9. MULTIPLE STEP KEY

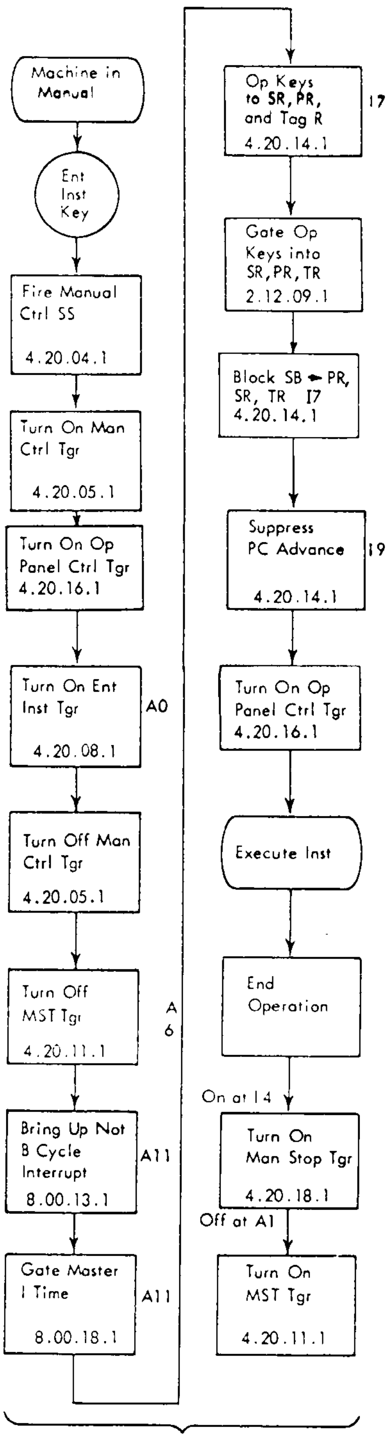

Machine in Manual
Ent Inst Key
Fire Manual Ctrl SS 4.20.04.1
Turn On Man Ctrl Tgr 4.20.05.1
Turn On Op Panel Ctrl Tgr 4.20.16.1
Turn On Ent Inst Tgr 4.20.08.1
A0
Turn Off Man Ctrl Tgr 4.20.05.1
Turn Off MST Tgr 4.20.11.1
A 6
Bring Up Not B Cycle Interrupt 8.00.13.1
A11
Gate Master I Time 8.00.18.1
A11
Op Keys to SR, PR, and Tag R 4.20.14.1
17
Gate Op Keys into SR, PR, TR 2.12.09.1
Block SB ← PR, SR, TR I7 4.20.14.1
Suppress PC Advance 4.20.14.1
19
Turn On Op Panel Ctrl Tgr 4.20.16.1
Execute Inst
End Operation
On at I4
Turn On Man Stop Tgr 4.20.18.1
Off at A1
Turn On MST Tgr 4.20.11.1

FIGURE 6.1-11. ENTER INSTRUCTION

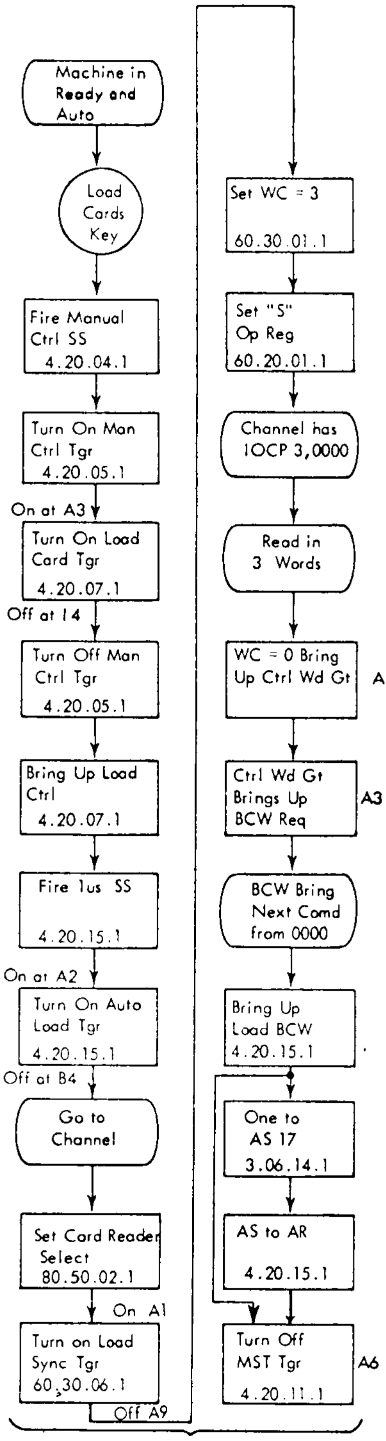

Machine in Ready and Auto
Load Cards Key
Fire Manual Ctrl SS 4.20.04.1
Turn On Man Ctrl Tgr 4.20.05.1
On at A3
Turn On Load Card Tgr 4.20.07.1
Off at I4
Turn Off Man Ctrl Tgr 4.20.05.1
Bring Up Load Ctrl 4.20.07.1
Fire Ius SS 4.20.15.1
On at A2
Turn On Auto Load Tgr 4.20.15.1
Off at B4
Go to Channel
Set Card Reader Select 80.50.02.1
On A1
Turn on Load Sync Tgr 60.30.06.1
Off A9
Set WC = 3 60.30.01.1
Set "S" Op Reg 60.20.01.1
Channel has IOCP 3,0000
Read in 3 Words
WC = 0 Bring Up Ctrl Wd Gt
A2
Ctrl Wd Gt Brings Up BCW Req
A3
BCW Bring Next Comd from 0000
Bring Up Load BCW 4.20.15.1
One to AS 17 3.06.14.1
AS to AR 4.20.15.1
Turn Off MST Tgr 4.20.11.1
A6

FIGURE 6.1-12. LOAD CARDS

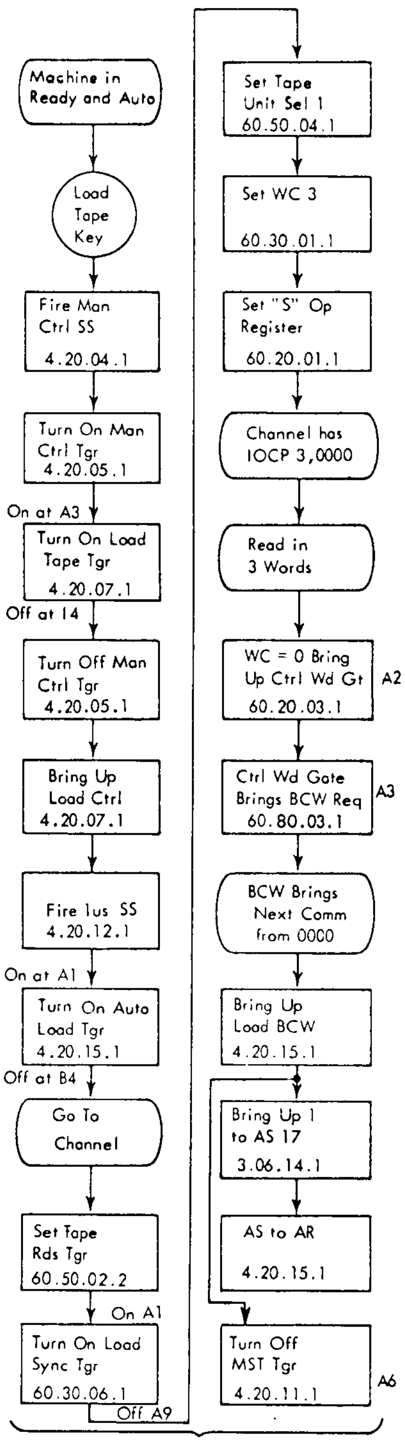

Machine in Ready and Auto
Load Tape Key
Fire Man Ctrl SS 4.20.04.1
Turn On Man Ctrl Tgr 4.20.05.1
On at A3
Turn On Load Tape Tgr 4.20.07.1
Off at I4
Turn Off Man Ctrl Tgr 4.20.05.1
Bring Up Load Ctrl 4.20.07.1
Fire Ius SS 4.20.12.1
On at A1
Turn On Auto Load Tgr 4.20.15.1
Off at B4
Go To Channel
Set Tape Rds Tgr 60.50.02.2
On A1
Turn On Load Sync Tgr 60.30.06.1
Off A9
Set Tape Unit Sel 1 60.50.04.1
Set WC 3 60.30.01.1
Set "S" Op Register 60.20.01.1
Channel has IOCP 3,0000
Read in 3 Words
WC = 0 Bring Up Ctrl Wd Gt 60.20.03.1
A2
Ctrl Wd Gate Brings BCW Req 60.80.03.1
A3
BCW Brings Next Comm from 0000
Bring Up Load BCW 4.20.15.1
Bring Up 1 to AS 17 3.06.14.1
AS to AR 4.20.15.1
Turn Off MST Tgr 4.20.11.1
A6

FIGURE 6.1-13. LOAD TAPE

card enter storage location zero, one, and two. The word entered in address zero should be a control word. When the word counter goes to zero, the channel asks for another control word from location zero. The CPU gets its next instruction from location one.

Load Tape Key (Figure 6.1-13)

see p 811 A

~~The load tape key works the same as the load cards key, causing a reset to the CPU and all channels not in manual status. The load tape key also causes a read select of tape unit 1 on channel A, IOCP word count of 3. The next command for the channel will come from location zero, and the next instruction for CPU will come from location 00001.~~

Sense Control

There are two types of sense controls on the operator's panel: sense lights and sense switches. The conditions of these sense devices may be checked by machine instructions and used to control program flow.

The four sense lamps on the panel may be turned on or off by instructions in the main program and then checked by the sense instructions. The condition of the lamp determines if the program steps are to be skipped.

The six sense switches may be set on or off from the operator's panel. The condition of the switch may then be checked by sense instructions in the program to determine whether to skip the following program step. Skips when switch is down, not skip when switch is up.

6.2.00 CUSTOMER ENGINEER'S TEST PANEL

In addition to the indicators and manual controls on the operator's panel, the customer engineer has at his disposal the indicators and controls on the customer engineer's test panel.

The indicators provide a means for checking the address register contents, the tally counter, various test triggers, and the cycle time. The switches and jacks provide means to continually execute any instruction, control I-O operation, control B time, and step through instructions cycle by cycle.

Figure 6.2-1 shows the layout of indicators and controls for this panel and designates system page locations.

6.2.01 Indicators

Indicators on the customer engineer's test panel are of the incandescent type. Indicators connected to test triggers are on when the trigger is on and off when the trigger is off. Indicators concerned with the registers and counters signify a 1 when they are on and a 0 when off.

Address Register

The address register contains the address portion of the instruction under execution.

New

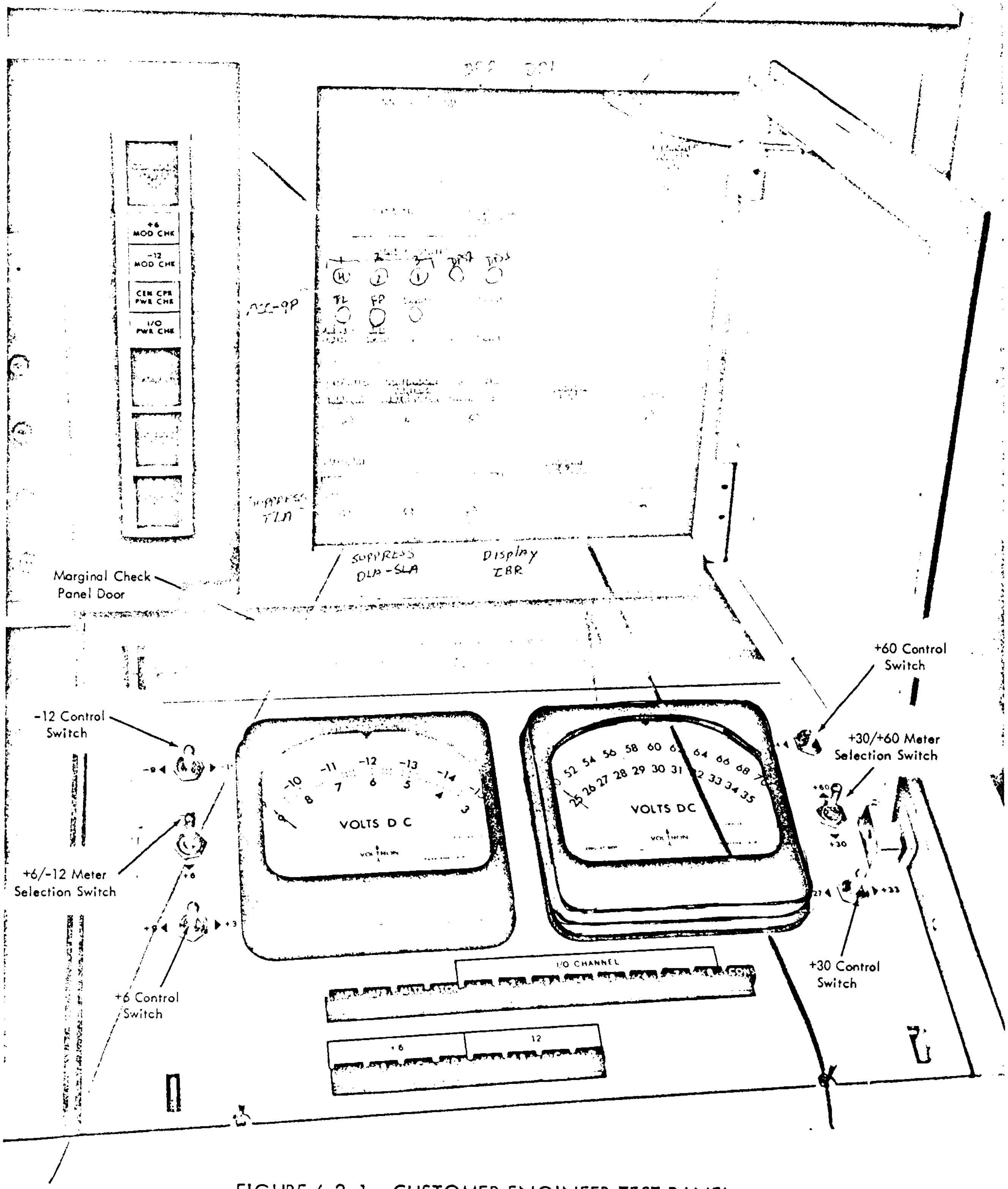


FIGURE 6.2-1. CUSTOMER ENGINEER TEST PANEL

Cycle Time

The cycle time indicators indicate in which cycle the machine is currently operating: I, E, L, or B time. Also, in connection with cycle time, is the "multiple error" indicator. This lamp is on whenever the machine is trying to perform two different cycles at the same time other than an L cycle and a B cycle, which is a normal share operation. Refer to Systems 08.00.17.1

Tally Counter

The tally counter differentiates between the L cycles of floating point instructions and provides gating for their different operational steps. The tally counter is divided into five steps. The lamps on the test panel indicate in which of the five steps the machine is currently operating. Refer to Systems 02.10.21.1 and 02.10.22.1.

T-2

Indicator T-2 signifies the condition of the T-2 trigger in floating add or subtract, multiply, and floating divide. During floating add or subtract, T-2 is used to indicate whether or not the multiplier quotient equals 0. During floating divide, T-2 is turned on if the quotient is greater than 2. T-2 is turned on at the beginning of a floating multiply second step to gate second step operations. The T-2 trigger is found on Systems 2.10.38.1.

FP

The floating point (FP) lamp indicates the condition of the FP trigger on Systems 2.10.29.1. The trigger is used to store certain conditions throughout the floating point operations. The conditions that turn on this trigger and indicator are shown in Figure 6.2-2.

9 Carry

This lamp indicates the condition of the 9 carry trigger on Systems 2.10.37.1. The trigger is turned on whenever there is a carry from adder position 9.

9 Overflow

The 9 overflow lamp signifies the condition of the 9 overflow trigger on Systems 2.10.39.1. The trigger and lamp come on whenever AC 9=1 during an accumulator left shift, or during a multiply add cycle when there is an adder 9 carry.

Q Carry

The Q carry lamp indicates the condition of the Q carry trigger on Systems 2.10.36.1. This trigger is turned on whenever there is a carry out of adder position Q.

Master Stop

The master stop lamp is turned on whenever the master stop trigger on Systems 4.20.11.1 is turned on.

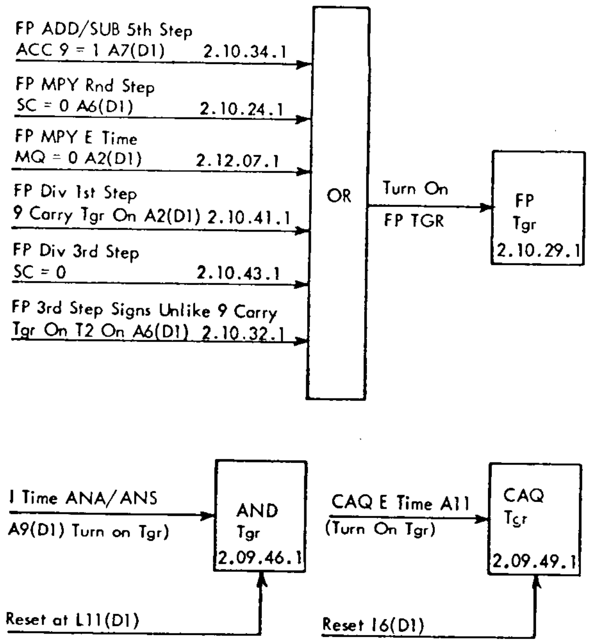


FIGURE 6.2-2. AND, CAQ, AND FP TRIGGERS

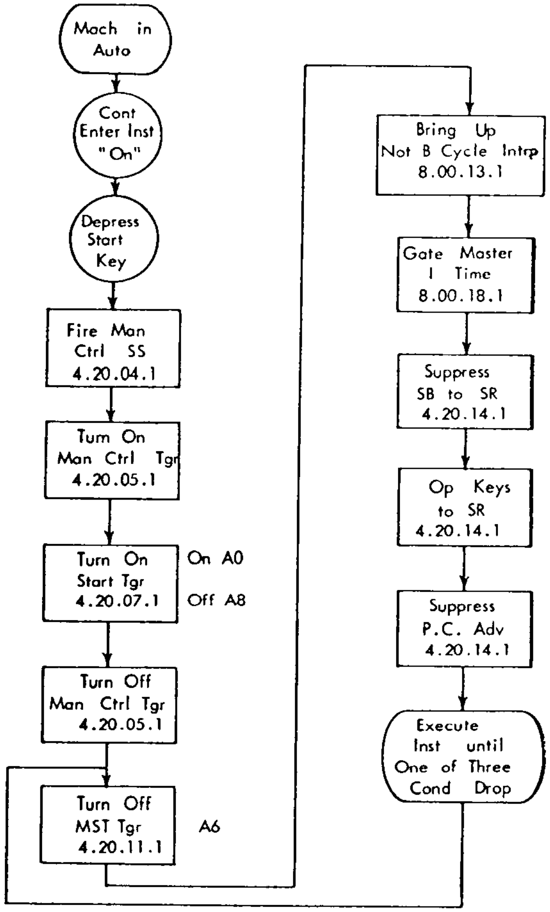


FIGURE 6.2-3. CONTINUOUS ENTER INSTRUCTION

End Operation

The end operation indicator is turned on whenever the end operation trigger on Systems 8.00.09.1 is turned on.

AND

The AND lamp indicates the condition of the AND trigger on Systems 2.09.46.1. It can be used to distinguish between the first and second E cycles of an ANS operation. It is on during the first E cycle of an ANS or ANA and off during the second E cycle of an ANS. See Figure 6.2-2.

CAQ

The CAQ indicator signifies the condition of the CAQ trigger on Systems 2.09.49.1. It denotes the difference between the first and succeeding E cycles of a CAQ instruction. The trigger is off for the first E cycle and on for the remaining E cycles and I time of next instruction. See Figure 6.2-2.

X Carry

The X carry lamp indicates the condition of the X carry trigger on Systems 2.12.76.1. The trigger is on when the machine is not in memory nullification mode and a carry occurs from adder position 3. If the machine is in memory nullification mode, a carry from adder position 4, when in 16K mode, or a carry from adder position 5, when in 24K mode, turns on the trigger.

~~6.2.02~~ Switches

I-O Interlock Switch

The I-O interlock switch is effective only when the system is in manual status. It has two positions: automatic and manual. With both the automatic-manual and I-O interlock switches set in the manual position, the computer will stop after executing each instruction. With the I-O interlock switch set to automatic, the machine will not stop if an I-O device is in operation. The normal setting (automatic) of the I-O interlock switch allows the computer to continue at high speed after I-O selection, to allow normal operation of the I-O device. The machine will stop after each instruction, providing no I-O device or data channel is in use.

Continuous Enter Instruction (Figure 6.2-3)

The continuous enter instruction switch is effective in automatic or manual status. With this switch on, all instructions are obtained from the operator's panel entry keys, not memory. The instruction counter is not used and does not advance. The IC contents will not be altered unless a skip, trap, or transfer results from the instruction in the entry keys. If the system is in automatic status, the instruction will be executed at normal operating speeds; if in manual status, the speed will be under control of multistep or single-step operations.

Multiple Step High and Low Speed

high speed - 50 ± 10 instructions/secon
low speed - 10 ± 2 instructions/secon

The following explanation is with reference to Section 3.12.00. With this switch in the high position, the multiple step key on the operator's panel performs program steps, one after another, every 24 ms. With the switch in the low position, program steps are performed every 104 ms. The high-speed position is advantageous when stepping through index loops.

B Cycle Controls

These switches allow the machine to operate normally when they are in the down position. In the up position, they modify normal operation as follows:

Interrupt:	Prevents the 7606 from obtaining a B cycle by interrupting a CPU instruction.
Share:	Prevents the 7606 access to core storage during CPU L cycles.
End Operation:	Prevents the 7606 from taking a B cycle when a CPU instruction ends operation.

The switches can be used if the customer engineer wants to test the end operation procedure of obtaining a B cycle. The interrupt and share switches would then be put in the up position. This prevents a B cycle from being started in any manner except end operation. Any of these switches in the up position turns off the ready light.

Machine Cycle Key (Figure 6.2-4)

The machine cycle receptacle on the customer engineer's test panel accepts the machine cycle key. This key is used to sequentially step through the basic machine cycles I, E, and L. When the machine cycle key is inserted into the customer engineer's panel and pressed, the machine cycle trigger and machine cycle gate trigger are turned on at A-0. The machine cycle trigger turns off the MST and allows for the proper cycle time, after which the MST is turned on again. The machine cycle gate does two things. First, it allows only three shifts per cycle on a shift operation, and forces a wait until the shift counter equals zero before ending operation in L time. Second, it is used to hold up "manual I time cntl" to insure that the machine will use a full I cycle.

"Manual I time cntl" is AND'ed with the output of the master I time trigger (Systems 08.00.18.1). Thus, if "manual I time cntl" drops, I time is finished. Usually, turning on the MST at I7 causes "manual I time cntl" to fall and stop I time at I7. This allows the machine to finish with an instruction before the following instruction is brought in. Normally, the go to I time trigger (Systems 08.00.12.1) is turned off at I9, but because there is no I9, it remains on. The all pulse is blocked from resetting the master I time trigger (Systems 08.00.08.1) by "minus on not go to I time." Therefore, if we start again, the MST is turned off at A6, bringing up "manual I time cntl" and starting the cycle at I6.

To insure getting a full I cycle when using the machine cycle key, the machine cycle gate trigger holds up "manual I time cntl" in the OR circuit on Systems 04.20.10.1.

When the key is not in use, a plug (sent with the system) that shorts pins 1 and 3 must be inserted.

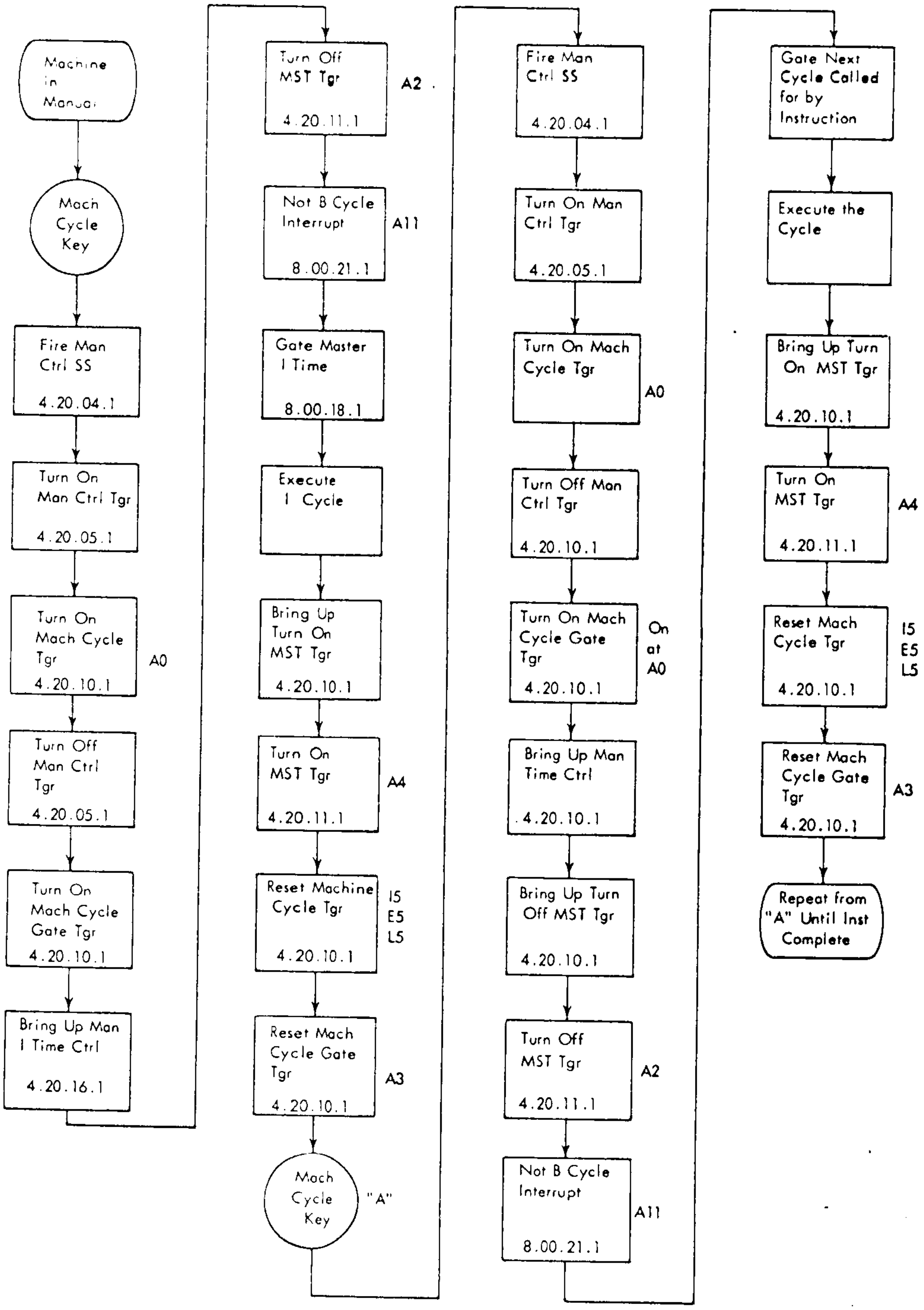

Machine in Manual
Mach Cycle Key
Fire Man Ctrl SS 4.20.04.1
Turn On Man Ctrl Tgr 4.20.05.1
Turn On Mach Cycle Tgr 4.20.10.1
A0
Turn Off Man Ctrl Tgr 4.20.05.1
Turn On Mach Cycle Gate Tgr 4.20.10.1
Bring Up Man I Time Ctrl 4.20.16.1
Turn Off MST Tgr 4.20.11.1
A2
Not B Cycle Interrupt 8.00.21.1
A11
Gate Master I Time 8.00.18.1
Execute I Cycle
Bring Up Turn On MST Tgr 4.20.10.1
Turn On MST Tgr 4.20.11.1
A4
Reset Machine Cycle Tgr 4.20.10.1
I5
E5
L5
Reset Mach Cycle Gate Tgr 4.20.10.1
A3
Mach Cycle Key
"A"
Fire Man Ctrl SS 4.20.04.1
Turn On Man Ctrl Tgr 4.20.05.1
Turn On Mach Cycle Tgr
A0
Turn Off Man Ctrl Tgr 4.20.10.1
Turn On Mach Cycle Gate Tgr 4.20.10.1
On at A0
Bring Up Man Time Ctrl 4.20.10.1
Bring Up Turn Off MST Tgr 4.20.10.1
Turn Off MST Tgr 4.20.11.1
A2
Not B Cycle Interrupt 8.00.21.1
A11
Gate Next Cycle Called for by Instruction
Execute the Cycle
Bring Up Turn On MST Tgr 4.20.10.1
Turn On MST Tgr 4.20.11.1
A4
Reset Mach Cycle Tgr 4.20.10.1
I5
E5
L5
Reset Mach Cycle Gate Tgr 4.20.10.1
A3
Repeat from "A" Until Inst Complete

FIGURE 6.2-4. MACHINE CYCLE KEY

SUPPRESS TLA -- the normal position of this switch is off or down. When on, this swithc turns off the ready light and suppresses executing TRA, TTR and the six index transfer instructions in overlapped fashion from the IBR (TLA).

SUPPRESS DLA/SLA -- the normal position of this switch is off or down. When on, this switch turns off the ready light and suppresses the data lookahead and store lookahead features, thus causing all instructions to execute full 2 microsecond I and E cycles.

DISPLAY IBR -- depressing this key replaces the contents of the storage register with the contents of the IBR. Effective in manual status only.

MEMORY DIAGNOSTIC MODE -- the normal position of this switch is off or down. When on, this switch turns off the ready light, suppresses all overlap functions and interchanges the high and low order bits between the remainder of the 7094 system and the 7302 core storage unit.

Switch Off	Switch On
AR 3 gates to MAR 17	AR 3 gates to MAR 3
AR 17 gates to MAR 3	AR 17 gates to MAR 17
Planes 0 - 35 contain all even-numbered addresses	Planes 0 - 35 contain all addresses below $40,000_8$
Planes 36-71 contain all odd-numbered addresses	Planes 36 -71 contain addresses above $37,777_8$

MACHINE CYCLE JACK -- when the ~~[illegible]~~ machine cycle key is inserted in this plug, depressing the key onee causes the machine to execute one and only one cycle. When the machine cycle key is not plugged in, a plug shorting pins 1 and 3 of the jack must be inserted.

AUXILIARY START AND RESET JACK-- whenx the auxiliary start and reset buttons are plugged into this jack they operate the same as sthe operator's panel start and reset buttons. (cont'd on p 822)

Auxiliary Start and Reset (cont'd)

~~This jack accepts the auxiliary start and reset key~~. The key is on a long cable, giving the customer engineer a means for starting and resetting the machine at points other than the console.

Phone Jack

The phone jack, in conjunction with the phone jacks on the other units in the system, provides a means of communication between customer engineers working at different units.

16K/24K Mode Switch

This switch is used with the compatibility package. When it is in the 16K position, 16K core storage positions are available to the 704 program; in the 24K position, 8K core storage positions are available to the 704 program.

DC On

The DC on switch controls the 400-cycle power supplied only to the 7151-II. Putting the DC on switch in the off position will immediately remove all power to the console, except the convenience outlets and the reset motor (operator's keys). All voltages should be normal about ten seconds after turning this switch to the on position.

6.3.00 MARGINAL CHECK

The system biasing network can be useful to the customer engineer. The controls are ocated on the marginal check console on the 7151. Any single gate or combination of tes in any single module, or combination of modules, can be biased at the same time. only exception is the 7302 memory. This module is under control of only one key, A, the whole module will be biased when this key is pressed.

ch module has a key for selecting it, and each gate has a key with key A for gate r gate B, etc.

biasing, all gates must be selected prior to varying the voltage. After the been varied, another gate cannot be selected until returning to normal volt- e A is being biased and it is decided to bias gate B instead, gate A MC relay opped out; therefore it will be necessary to take the MC voltage to normal cting gate B. If this is not done, gate A MC relay will have a hold through elay points. This means that both gates A and B would be biased instead of as desired.

possible to vary the +30v and +60v in the 7302. These controls are also e 7151, on the MC panel.

ying the different voltages, there are two meters to monitor the amount of g varied. One meter is for the +30 and +60 in the 7302, and one meter, for . -12 for the rest of the system. The meters indicate what the voltage is if relays are picked and the points are properly adjusted. A periodic check of the s, while biasing, should be made.