Ford

212 REFERENCE MANUAL

Technical Computer Services
Engineering Staff
December, 1968

PREFACE

This manual describes the Philco 212 Electronic Data Processing System, a member of the Philco 2000 series. Included in the manual is a description of the organization of the Philco 212 Central Processor and its associated registers. Each of the instructions performed by the Philco 212 Central Processor is described in detail. Detailed descriptions of the input-output units associated with the Philco 212 are contained in the Input-Output Systems Manual, TM-16.

This manual incorporates the changes to the previous edition of the manual, TM-29, announced in Philco Computer Users Memo 27-63.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

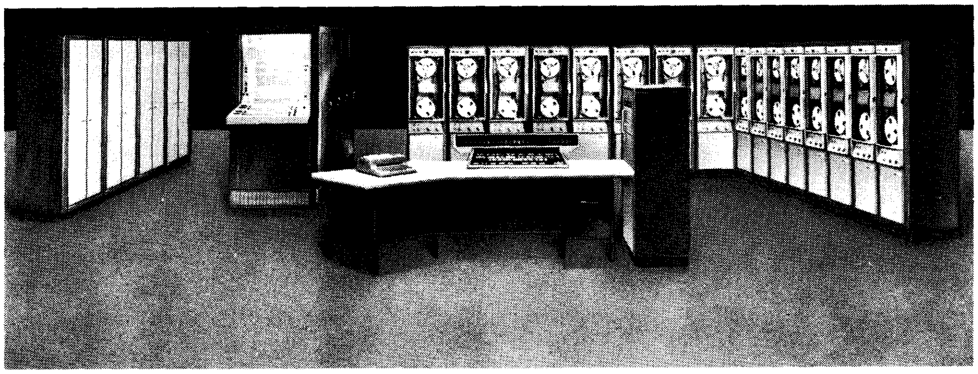
TABLE OF CONTENTS (Continued)

Chapter 1

PHILCO 212 DATA PROCESSING SYSTEM

INTRODUCTION

The Philco 212 Data Processing System is an extremely high-speed, large-scale electronic computer designed for business, scientific and real-time operations.

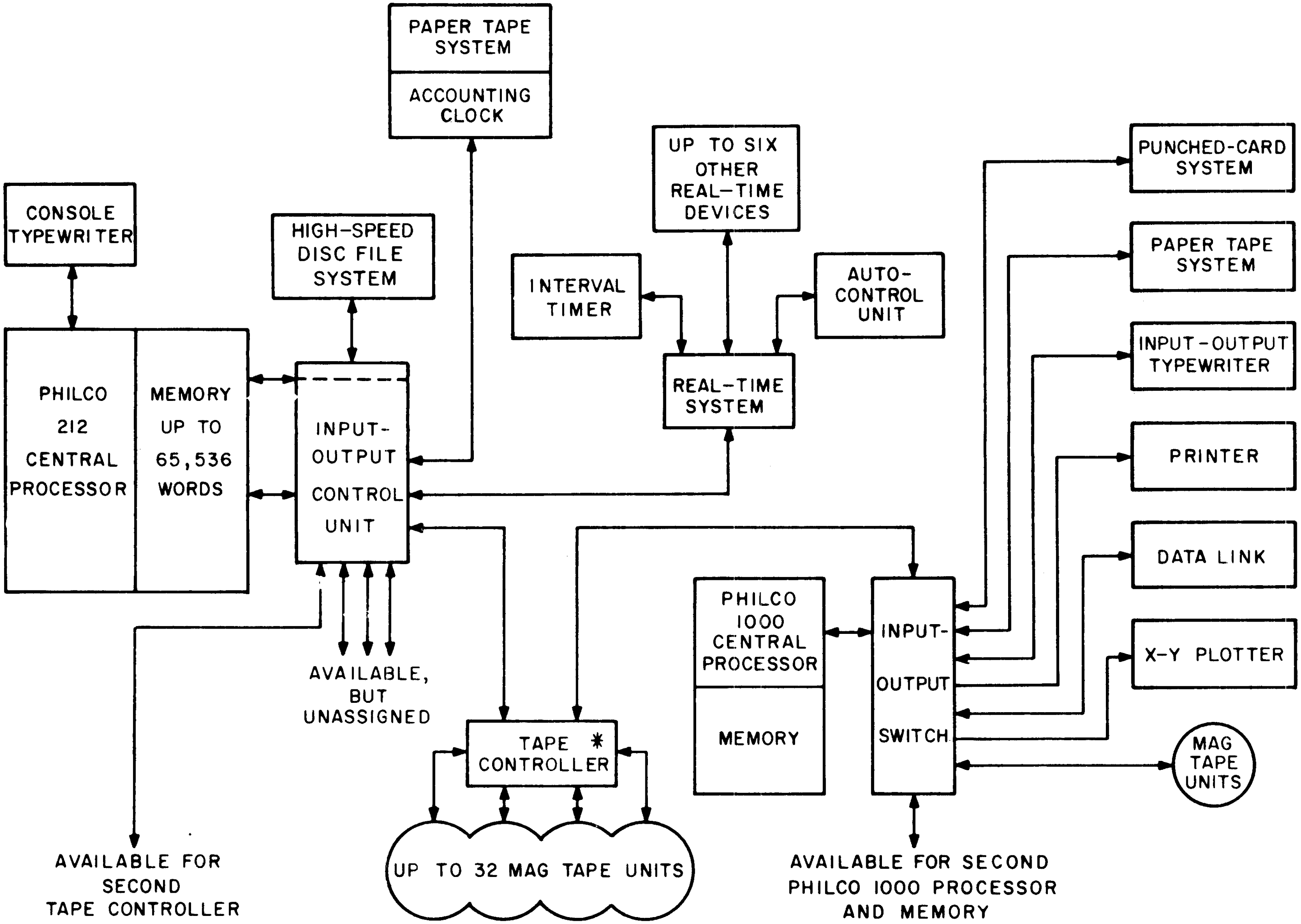


Philco 212 System

An Input-Output Control Unit connects eight input-output channels to the Philco 212 Central Processor. Direct input or output is made on 240,000 or 90,000 six-bit characters per second magnetic tape. Real-time input and program interrupt are provided by the Real-Time System. Any computer in the Philco 1000 series can be connected to the magnetic tape system for input-output between the Philco 212 and one of the following units, or between any two of the following units.

- 1000 character per second Paper Tape Reader
- 60 character per second Paper Tape Punch
- 2000 or 600 card per minute Punched-Card Reader
- 100 or 200 card per minute Card Punch
- 300 or 900 line per minute Printer
- X-Y Plotter
- 240,000, 90,000, 25,020, and 9,000 Characters Per Second Magnetic Tape Units

The Philco 1000 relieves the Philco 212 of routine data handling functions such as input formatting and verification, file searching, conversion of punched card information to tape, and editing of output for the printer.


PAPER TAPE SYSTEM
ACCOUNTING CLOCK
CONSOLE TYPEWRITER
HIGH-SPEED DISC FILE SYSTEM
UP TO SIX OTHER REAL-TIME DEVICES
INTERVAL TIMER
AUTO-CONTROL UNIT
REAL-TIME SYSTEM
PUNCHED-CARD SYSTEM
PAPER TAPE SYSTEM
INPUT-OUTPUT TYPEWRITER
PRINTER
DATA LINK
X-Y PLOTTER
MAG TAPE UNITS
PHILCO 212 CENTRAL PROCESSOR
MEMORY UP TO 65,536 WORDS
INPUT-OUTPUT CONTROL UNIT
AVAILABLE, BUT UNASSIGNED
AVAILABLE FOR SECOND TAPE CONTROLLER
TAPE CONTROLLER *
UP TO 32 MAG TAPE UNITS
PHILCO 1000 CENTRAL PROCESSOR
MEMORY
INPUT-OUTPUT SWITCH
AVAILABLE FOR SECOND PHILCO 1000 PROCESSOR AND MEMORY

* AN INPUT-OUTPUT PROCESSOR CONNECTING UP TO 16 TAPES MAY BE USED INSTEAD OF A TAPE CONTROLLER

Figure 1. Block Diagram of a 212 System

The main magnetic core storage for the Philco 212 has a memory access time of 0.9 microseconds and a capacity of up to 65,536 words. Auxiliary storage of 5,242,880 words is provided by the Disc File System.

The Philco 212 is compatible with other computers in the Philco 2000 series. Programs and routines used on the Philco 210 and 211 can be run on the 212 without reprogramming. Ease of programming is assured by the availability of TAC, ALTAC and COBOL compilers.

Instruction and operand look ahead permits as many as seven instructions to be processed simultaneously. The Philco 212 features asynchronous processing between instructions, as well as within instructions. For example, while the computer is storing or waiting to store the result of an operation, it is executing the next instruction, indexing and accessing operands for still another, and accessing the next four instructions from memory.

Under normal conditions, memory access time becomes negligible because access time for instructions and operands is generally overlapped by the arithmetic execution time of the preceding arithmetic instruction.

Using the 1.5 microsecond memory, typical rates of the arithmetic operations, including instruction and operand access, expressed in operations per second are as follows (the instructions to which these rates apply are indicated in parentheses):

	Fixed Point		Floating Point	
Addition and Subtraction	(AD,SD)	1,680,670	(FAD,FSD)	626,950
Multiplication	(MA)	151,630	(FMA)	199,400
Division	(DAQ)*	60,420	(FDAQ)*	79,680

CENTRAL PROCESSOR

The Central Processor consists of magnetic core memory and a Control Section. The Central Processor processes data stored in magnetic core memory in accordance with the interpretation of the program instructions. It is the function of the Control Section to interpret and execute the instructions.

Control Section

The Philco 212 normally executes instructions in sequential order. Each memory location can contain two instructions which are transferred simultaneously to the Program Register in the Instruction Unit (see page 19).

The Control Section of the Philco 212 system executes 250 instructions, including 59 floating-point instructions. In addition, input-output orders for each particular type of device are available for executing input-output orders and for checking for transmission, the amount transmitted, and possible transmissio errors.

* These instructions have a double-length dividend.

Magnetic Core Storage System

The Magnetic Core Storage System is a random access device capable of storing data in banks of 8192 words, 48 binary digits (plus 8 parity bits) in length, and has an average memory access time of 0.9 microsecond. A complete read-write cycle is performed in 1.5 microseconds. Magnetic Core Storage Systems are available with a total storage of 32,768 (32K) or 65,536 (65K) words.

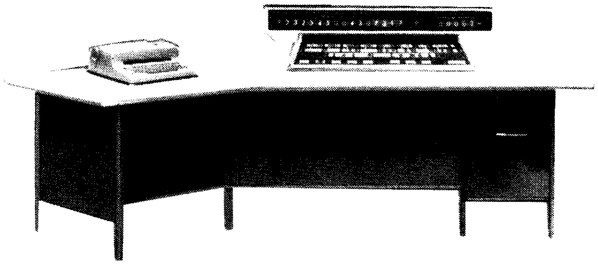
When the Central Processor has transferred information to registers of a memory unit, the Processor proceeds to other operations while memory is independently completing its read-write cycle.

Separate banks of 8192 words can be accessed virtually simultaneously. Therefore, instruction access, operand access, result storage, and input-output operations can proceed concurrently if the values are in separate banks.

Data transferred to or from memory is checked for an odd parity read. If a parity error is detected, the error is indicated on the Operator's Console and the Central Processor halts (see Appendix D).

OPERATOR'S CONSOLE AND CONSOLE TYPEWRITER

The Operator's Console provides indicators and manual controls for monitoring the operations of the Philco 212.



Operator's Console and Console Typewriter

By means of the Console Typewriter, the operator has direct and immediate access to the magnetic core storage. Input can be typed from the keyboard and a typed copy is prepared for checking purposes. Output can be typed at a speed of 15 characters per second. A 16 character Console Typewriter Buffer allows the Central Processor to proceed without waiting for the completion of each type-out cycle.

INPUT-OUTPUT CONTROL UNIT

The Input-Output Control Unit is a connecting unit between eight input-output channels and two lines to memory. The input-output units available for these channels are discussed below.

DISC FILE SYSTEM

The Philco Disc File System provides a high-speed, rapid access, auxiliary storage for the Philco 212. Data or programs not required immediately in memory, but which must be available in milliseconds, are stored in the Disc File System. When the Central Processor requests this data, it is transmitted to the magnetic core memory, where it can be accessed by Central Processor instructions.

The Disc File System stores 41,943,040 alphanumeric characters (5,242,880 words).

The Disc File System transfers 960,000 alphanumeric characters per second (120,000 words per second). Operation may be continuous for up to 32,768 words. Parity is checked for each character transmitted or received.

The Disc File System operates through its own section of the Input-Output Control Unit and its own line to memory. Data transmission to and from the Disc File System proceeds simultaneously with computation and other input-output operations.

MAGNETIC TAPE SYSTEMS

Two magnetic tape systems are available for the Philco 212:the High Performance 240,000 Characters Per Second Magnetic Tape System and the 90,000 Characters Per Second Magnetic Tape System.

High Performance Magnetic Tape System

The primary input-output medium of the Philco 212 is the High Performance Magnetic Tape System. Each magnetic tape unit in the system operates at a peak transfer rate of 240,000 six-bit characters per second.

The Magnetic Tape System consists of one or two Tape Controllers, each of which provides two or four channels (Data Controllers) through which data can pass between memory and associated magnetic tape units. Up to 32 magnetic tape units can be connected to each Tape Controller. Any tape unit requested is automatically assigned to any available Data Controller within that Tape Controller, thus providing for up to four simultaneous read and/or write operations per Tape Controller.

Tape stations accommodate reels with 3600 feet of one-inch wide magnetic tape. Each reel contains up to 72.0 million 6-bit characters or 9.0 million Philco 48-bit words when data is recorded in a record size of 4096 words. The record size may vary from one to 4096 words (8 to 32,768 characters).

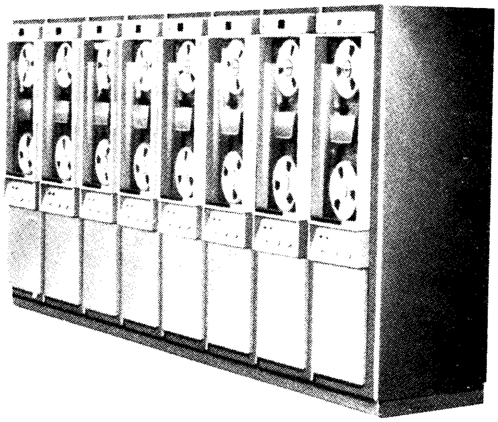
Up to 16 records, of up to 32,768 6-bit characters each, can be read or written by one input-output order with no start-stop time between records. Start-stop time is 2.5 milliseconds.

Tapes may be written forward, and read forward or reverse. Accuracy of reading and recording is assured by parity checks, word counts, and the use of separate read/write heads. While information is being recorded, it is read back and checked for validity. When an error is detected, the block is automatically rewritten. Similarly, when an error is detected during reading, the automatic verification feature rereads the block. A write-disable feature is provided to prevent the unintentional erasure of the magnetic tape.

90KC Magnetic Tape System

For Philco 212 installations that do not require the High Performance Magnetic Tape System, the 90,000 Character Per Second (90KC) Tape System is available.

Up to 16 tapes with a peak transfer rate of 90,000 6-bit characters per second can be used, on line, by connecting them directly to an Input-Output Processor. Up to four read and/or write operations can take place simultaneously.



Magnetic Tape Units

The one-inch magnetic tape used is available in lengths up to 3600 feet. Each reel contains up to 19 million 6-bit characters or 2.4 million Philco 48-bit words. Data is recorded in fixed records of 128 words. Up to 16 records may be read or written by one input-output order with no start-stop time between records. Start-stop time is 2.5 milliseconds.

Tapes may be written forward, or read forward or reverse. The 90KC Tape System has the same checking features as the High Performance Magnetic Tape System.

Input-Output transmission orders for the 90KC Tape System are compatible with the High Performance Magnetic Tape System.

ACCOUNTING CLOCK SYSTEM

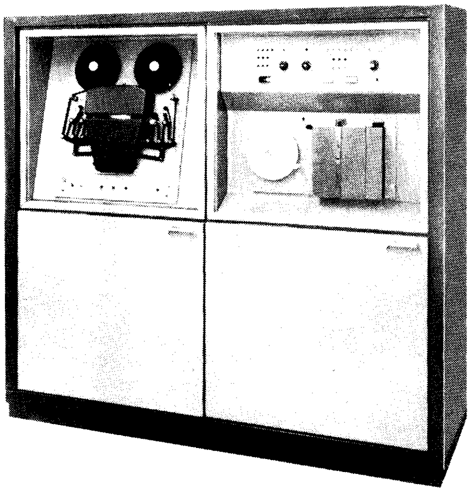
The Accounting Clock System is used to transmit the date (month and day) and the time of the day (hours, minutes and tenths of minutes) to memory whenever the program calls for this information.

The Accounting Clock System contains switches for the initial clock setting and automatically corrects for the lengths of months. A manual switch is used to indicate the extra day in a leap year.

The Accounting Clock System shares access to the Central Processor with the On-Line Paper Tape System. A special interface is required to connect the Accounting Clock System to the Input-Output Control Unit.

ON-LINE PAPER TAPE SYSTEM

The On-Line Paper Tape System reads or punches paper tape as input to or output from the Central Processor through the Paper Tape/Accounting Clock channel. Data in the form of five- or seven-level punched paper tape may be photoelectrically read directly into the Philco 212 at the rate of 1000 characters per second. With the Paper Tape Punch, data may be punched in five or seven levels onto paper tape at the rate of 60 characters per second. A special interface is required to connect the Paper Tape System to the Input-Output Control Unit.



Paper Tape System

REAL-TIME SYSTEM

The Real-Time System of the Philco 212 Electronic Data Processing System provides a channel to the Central Processor for high-priority data. A special interface is required to connect the Real Time System to the Input-Output Control Unit.

A Real-Time Scanner in the Real-Time System transfers information between the Central Processor and a real-time, such as the Interval Timer or the Auto-Control Unit.

Models of the Real-Time System are available to multiplex and check one, four or eight real-time units.

The Scanner sequentially interrogates each real-time unit that is connected to it. When a real-time unit is ready to transmit information, the Real-Time System generates a signal for the Central Processor. The signal sent to the Central Processor may cause an automatic interrupt (see Auto-Control Unit below).

Auto-Control Unit

The Auto-Control Unit provides the means of interrupting a program whenever specified conditions appear in the Central Processor or in the input-output system.

The interrupt feature is under program control, the programmer specifying which of up to 48 preselected system conditions shall be allowed to interrupt the Central Processor, and what action is to be taken when these conditions are present.

When any one of the designated signals is received, the contents of significant registers are stored and control is automatically transferred to a predesignated memory location. Once processing appropriate to the interrupt signal is completed the interrupt program may use the stored registers to return to the previously interrupted program.

Interval Timer

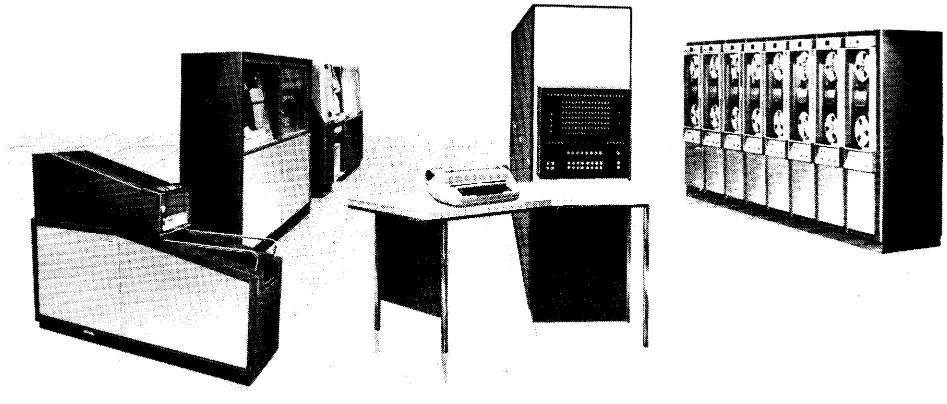
The Interval Timer transmits through the Real-Time System and times, in milliseconds, intervals of up to 9.32 hours.

The Timer is an electronic ''alarm clock'' which can provide an interrupt signal through the Auto-Control Unit when a designated period of time has elapsed. The Timer must be preset by a program to contain the desired time lapse. The Interval Timer Unit is program addressable at all times and maybe inspected without affecting its contents or function.

PHILCO 1000 COMPUTER SERIES

The Philco 1000 Computer Series provides for input formatting and verification, file searching, conversion of punched card information to tape, and editing of output for the printer. The Philco 1000 may be connected to the Tape Controller or to the Input-Output Processor, sharing Magnetic Tape Units with the Philco 212, or operated off-line independent of the Philco 212.

A basic Philco 1000 system consists of a Processor, 4096 characters of Magnetic Core Memory, and an Input-Output Switch for the connection of input-output devices. An expanded system may have two separate Processors. Each of these Processors may have up to 32,768 characters of Magnetic Core Memory.



Philco 1000 System

One or both Processors share a common Input-Output Switch. The Input-Output Switch is available in four models. If one Central Processor is used, an Input-Output Switch is available to permit one transmission between the memory of that Processor and any one of 4 or 8 input-output channels. If two Central Processors are used, an Input-Output Switch is available to permit any two concurrent transmissions, one with each Central Processor's Memory Section, and the other with two of 4 or 8 input-output channels. Each input-output channel may handle several similar input-output units, the number depending on the type of device.

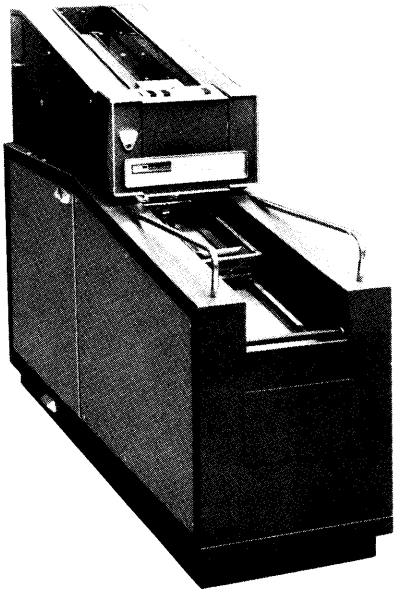
The input-output units which may be connected to the Philco 1000 include the Punched-Card System, Printing System, Magnetic Tapes, X-Y Plotter, Paper Tape System, and the Input-Output Typewriter. Input-Output Buffers are available for selected units.

A Data Link to provide high-speed data transmission between the Philco 212 Electronic Data Processing System and remote input-output stations may also be connected to the Philco 1000.

Each Processor has its own coincident-current magnetic core memory. Data is accessed serially, a character at a time, with a full memory cycle of less than 5 microseconds. Effective memory access time is 3 microseconds. Each character consists of six bits plus a parity bit and is checked each time the character is accessed. Core Memory is available in models of 8192, 16,384, and 32,768 characters.

High-Speed Punched-Card System

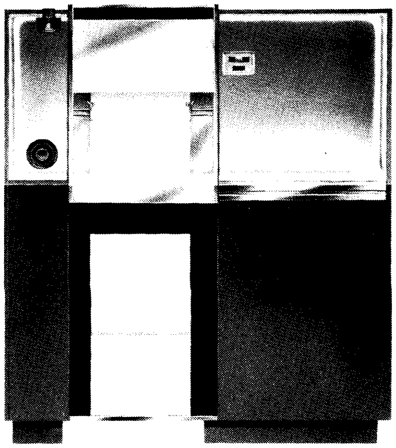
The Punched-Card System reads 80-column cards at the rate of 2000 or 600 cards per minute and punches them at a rate of 100 or 200 cards per minute. Cards may be read in Hollerith or binary modes.



High-Speed Card Reader

The Card Readers have dual read stations for complete checking of the data. The data read by the first read station is compared with the data read by the second read station. The Card Punches have a read after punch to provide for punching accuracy. Checking of the system components, parity, and card alignment insures the accuracy of the reading and punching operations.

Printers A 300 or a 900 line per minute Printer may be connected to the Philco 1000. Both Printers print 64 characters (see Appendix B) in 120 positions per line on continuous single or multi-part forms from 4 to 20 inches in width for the 300 LPM Printer and from 4 to 19 inches in width for the 900 LPM Printer.



High-Speed Line Printer

Magnetic Tapes Magnetic tape units with transfer rates of 240,000, 90,000, 25,020 or 9,000 characters per second may be connected to the Philco 1000. Magnetic Tape units are also available to read or write tapes prepared in IBM 729 format.

X-Y Plotter The X-Y Plotter plots, under program control, discrete points, continuous lines, curves, letters, numerals, and symbols. The information to be plotted is received from the Philco 1000. Two X-Y Plotters may be coupled to one channel of the Philco 1000 Input-Output Switch for concurrent operation. Data for Plotters on the same channel may be intermixed.

Continuous lines in both vertical (Y-axis) and horizontal (X-axis) directions can be plotted by the Plotter. Plotting along the Y-axis is done by vertically moving sprocketed, continuous feed paper on a bi-directional rotating drum. Plotting along the X-axis is performed by moving the pen horizontally across the plotting paper. Diagonal lines are drawn by combinations of X and Y movements; discrete points are made by raising, moving, and then lowering the pen.

Plotting is performed at a rate of 300 horizontal or vertical pen movements per second. One-hundred movements per inch are made at a rate of three inches per second. Pen movements up or down for points are performed at a rate of 10 per second.

The plotting area is 11 inches in width. The Plotter always steps 1/100 inch in either the X or Y direction. Points on a 45° diagonal are made by the plotter in a single step of 0.0141 inch. A plotted diagonal other than 45° must be made up of a series of pen carriage and paper drum movements.

Paper Tape System and Input-Output Typewriter

The specifications for the Paper Tape System and Input-Output Typewriter for the Philco 1000 are the same as the specifications for the Paper Tape System and Console Typewriter as described for the Philco 212, except that the Paper Tape System reads and punches five-, six-, seven- or eight-level paper tape.

Input-Output Buffers

The Philco 1000 Input-Output Buffers allow the Philco 1000 Central Processor to transfer data to or from selected input-output units without using time needed for data processing.

The Buffers have a capacity of 320 characters. The transfer rate between the Philco 1000 memory and a Buffer memory bank is a minimum of 200,000 characters per second. The transfer rate between an input-output unit and the Input-Output Buffer is determined by the input-output unit.

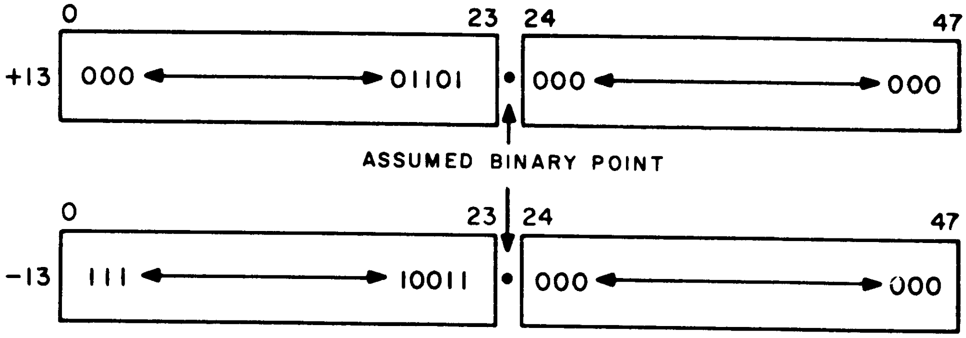
Chapter 2

THE PHILCO 212 WORD

The Philco 212 word comprises 48 data bits numbered, in this description, left to right from 0 through 47. Bit 47 is the least significant bit. For each six data bits there is an additional odd parity bit used for checking data transferred to or from memory. The parity bits are not shown in any diagrams in this manual.

FIXED-POINT DATA WORDS

During the execution of fixed-point operations (see Chapter 4), a word represents a binary value. A binary point, dividing the word into integral and fractional parts can be assumed to be any where within or outside the register. The first bit of the word, the sign bit, indicates whether the rest of the bits express a positive or negative value, zero indicating positive and one indicating negative. Negative numbers are represented in two's complement form.* For example, +13 and -13 appear as follows) the binary points are assumed to be between bits 23 and 24):



The computer subtracts by taking the two's complement of the subtrahend and then adding that two's complement to the minuend.

Fixed-Point Zero

Fixed-point zero is represented by a word of zeros.

Significant Bits

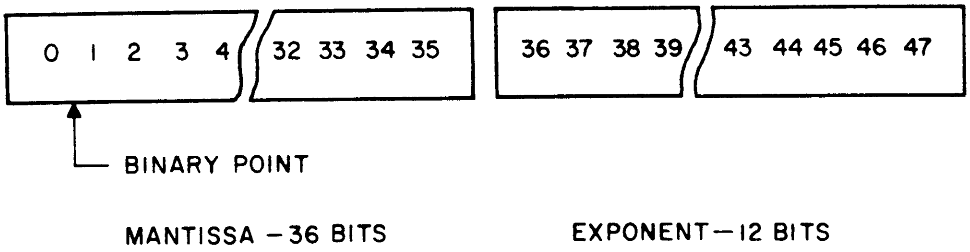
The first significant bit of a binary word is defined as the first bit following the sign bit which differs in setting from the sign bit. All subsequent bits are significant. Thus, in the examples above, the first significant bit is bit 20.

* A simple method of obtaining the two's complement of a binary number is to change all zeros to ones and all ones to zeros and then add one to the rightmost bit.

FLOATING-POINT DATA WORDS

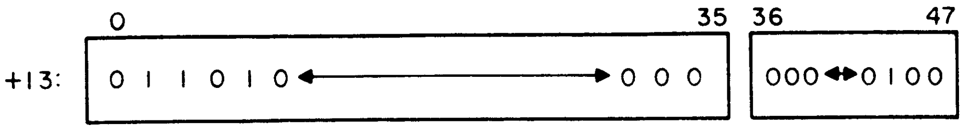
During the execution of floating-point arithmetic operations (see Chapter 5), all arithmetic registers are treated as if they were divided into two parts: a 36-bit mantissa and a 12-bit exponent to the base 2 as shown below.

The mantissa comprises the first 36 bits with bit zero as its sign bit; the exponent comprises bits 36-47 with its first bit (bit 36) as its sign bit.

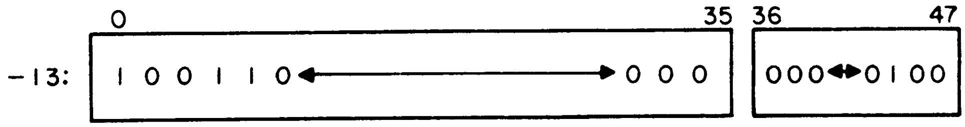


The mantissa is always fractional, with a binary point between bits zero and one. The exponent always represents an integral power of two. Both can be either positive or negative as indicated by their sign bits. Negative exponents and mantissas of negative numbers are represented in two's complement form.

For example, +13 and -13 appear as follows (both numbers are normalized, i.e., the first significant bit is bit one):



$\frac{13}{16} \times 2^4$



$-\frac{13}{16} \times 2^4$

Floating-Point Number Range

The largest representable mantissa is a zero in the sign position followed by 35 ones; the largest exponent is a zero followed by 11 ones. This number is equivalent to .9999999...x2^{2047}, which is very close to, but less than +1 x 2^{2047}. This value is equal to slightly more than 10^{616}.

Similarly, the algebraically smallest representable number is equal to minus one times the largest exponent, i.e., -1 x 2^{2047}. The normalized non-zero floating-point value which is smallest in absolute value is 0.5 x 2^{-2048} (or +1 x 2^{-2049}) which is slightly less than 10^{-617}.

The range of non-zero magnitude in floating-point representation, therefore, is from slightly more than 10^{616} to slightly less than 10^{-617}.

Floating-Point Zero

Floating-point zero is represented by a zero mantissa and an exponent with one in the sign position followed by zeros.

Double Precision Floating-Point Words

Double precision floating-point words are formed by two single length floating-point words with equal exponents (see Chapter 7).

BCD OR ALPHANUMERIC WORD

A BCD (Binary Coded Decimal) or alphanumeric word is composed of eight six-bit characters. Every group of six bits, beginning with bit zero, represents one of the Philco characters. (The Philco characters and their octal codes* are shown in Appendix B.) Input and output to certain peripheral equipment and some comparison operations are made in BCD form; internal arithmetic operations always assume fixed- or floating-point words as described above.

A BCD word with the octal codes representing the word PHILCO followed by two space characters (△△) is shown below.

0 5	6 11	12 17	18 23	24 29	30 35	36 41	42 47
47	30	31	43	23	46	60	60

INSTRUCTION WORD

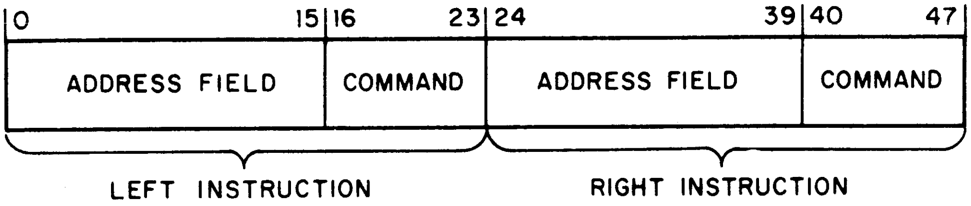
Two instructions, the left- and right-half instructions, comprise an instruction word. The normal sequence of executing instructions is first the left half and then the right half of one word, followed by the left-half and then the right-half instruction of the succeeding instruction word.

Each computer instruction contains 24 bits. The first 16 bits comprise the address field; the last 8 bits comprise the command field.

* Octal code uses a single digit to represent three bits, as follows:

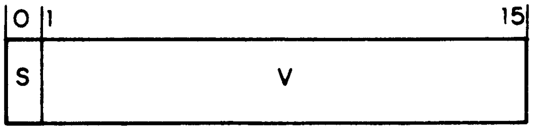
0 = 000	3 = 011	6 = 110
1 = 001	4 = 100	7 = 111
2 = 010	5 = 101	

Command Field The command field indicates the function to be performed* and the address field usually specifies the address of the operand to be used. The address field may be a memory address or some other value as required by the instruction. The format of an instruction word is shown in the diagram below:

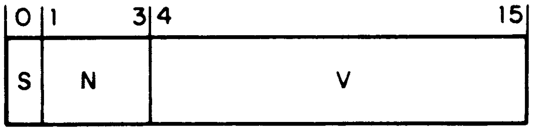


F-Bit The first bit of the command field is called the function bit, F. For arithmetic instructions, the F-bit specifies whether the arithmetic is to be performed in the fixed- (F-bit = 0) or floating-point (F-bit = 1) mode. For Jump Instructions (see page 53), the F-bit specifies whether a transfer of control is to be made to the left half of a location (F-bit = 0) or to the right half (F-bit = 1).

Address Field The address field is subdivided into an index register selector bit (S), a 3-bit index register field (N) to specify a particular index register, and a 12- to 15-bit variable field (V). If the S-bit is zero, no index register is used in forming an effective address field (see page 19) and the V-field is 15 bits. If the S-bit is one, an index register is used in forming an effective address field and the V-field is 12 bits.



ADDRESS FIELD IF S-BIT IS ZERO



ADDRESS FIELD IF S-BIT IS ONE

* In some cases, defining the full function also requires use of some bits in the address field.

On a 65K system, instructions having a full V-field (except Index Register Instructions, see page 63) use the value of the first bit of the 16-bit address of the memory location in which the instruction is stored as the value of the first bit in a 16-bit address, unless the instruction is extended by an EXT instruction immediately preceding it (see pages 88-92) or is controlled by an RPT or DR instruction (see pages 83-86). In the latter case, the values of the first bit of the 16-bit address of the memory location in which the RPT or DR instruction is stored is used as the first bit in the 16-bit address of the instruction controlled by the RPT or DR instruction.

For an instruction immediately following an EXT instruction, the V-field comprises up to 16 bits (enough to express the largest computer address); the S-bit and N-field used are taken from the address field of the EXT instruction.

INPUT-OUTPUT ORDER WORD

The TIO instruction (see page 52) causes pertinent fields of the word in the D Register (see page 21) to be transferred to the appropriate input-output system. This 48-bit word is known as an input-output order. Details on orders for specific input-output systems are contained in the Input-Output Systems Manual TM-16. The following is an example of an input order for the High Performance Magnetic Tape System.

0	12 15		18	19 23	24 35	36 39	40 47
	NRS		T C	UNIT	NWR	NRP	10100001

This instruction causes the Processor to space over 0 to 15 records as specified (NRS) which are any length from 1 to 4096 words (NWR), then to read 0-15 records (NRP) of the same length in the forward direction. If NRS and NRP are both zero, 16 records are read; if NWR is zero, 4096 word records are read.

Tape units 0 to 31 of either Tape Controller may be addressed by this order (UNIT); Tape Controller 0 or 1 may be designated by bit 18.

The TIO instruction itself contains the address of the memory location at which transmission begins.

Chapter 3

CONTROL SECTION OF THE PHILCO 212

The Control Section of the Philco 212 consists of an Instruction Unit, an Index Unit, an Arithmetic Unit, and a Store Unit.

INSTRUCTION UNIT

The Instruction Unit accesses memory instructions and stores up to four instructions until they can be accepted by the Index Unit. The Instruction Unit contains two Program Address Registers (PA and PA*) and two word-size Program Registers (PR and PR*), which allow a total of four instructions to be buffered in the unit. Controls in the Instruction Unit sequentially select the left half or right half of the word in one of the Program Registers for processing. The RPT and DR instructions (see pages 83 and 85) permit from one to four of the instructions stored in the Instruction Unit to be repeated without reaccessing memory for instructions.

INDEX UNIT

The Index Unit performs that part of the instruction which can be done prior to further execution which may be required in the Arithmetic Unit. Its major function is to obtain operands and to store them in the Operand Register until the Arithmetic Unit (see below) has completed the preceding instruction; thus, this function is performed while the Arithmetic Unit is processing the preceding instruction. Operand access therefore is overlapped by previous arithmetic operation, making operand access time a negligible factor in timing a program (see Appendix F).

Index Registers

Eight index registers are standard on the Philco 212. They are addressable memory-sized registers, i.e., they contain as many bits as are needed to express the largest computer address. They can receive information directly from an instruction or from the D Register of the Arithmetic Unit, and can transfer information to the JA and to the D Registers of the Arithmetic Unit.

Index registers are used as counters or to form the effective address of an instruction. Arithmetic performed on index registers is modulo memory size.

Effective Address

The effective address of an instruction is the final address of a memory location referenced by the instruction. The effective address may be directly specified by the V-field of the instruction or, if indexing is specified, may be formed as indicated by the table below.

C-Bits and Y-Bits

Every index register has an associated C-bit and Y-bit. Each of these bits may be set to either one or zero. Used in conjunction with each other, they indicate how the effective address of an instruction is to be formed, and how much the index register is to be incremented or decremented after forming the address.

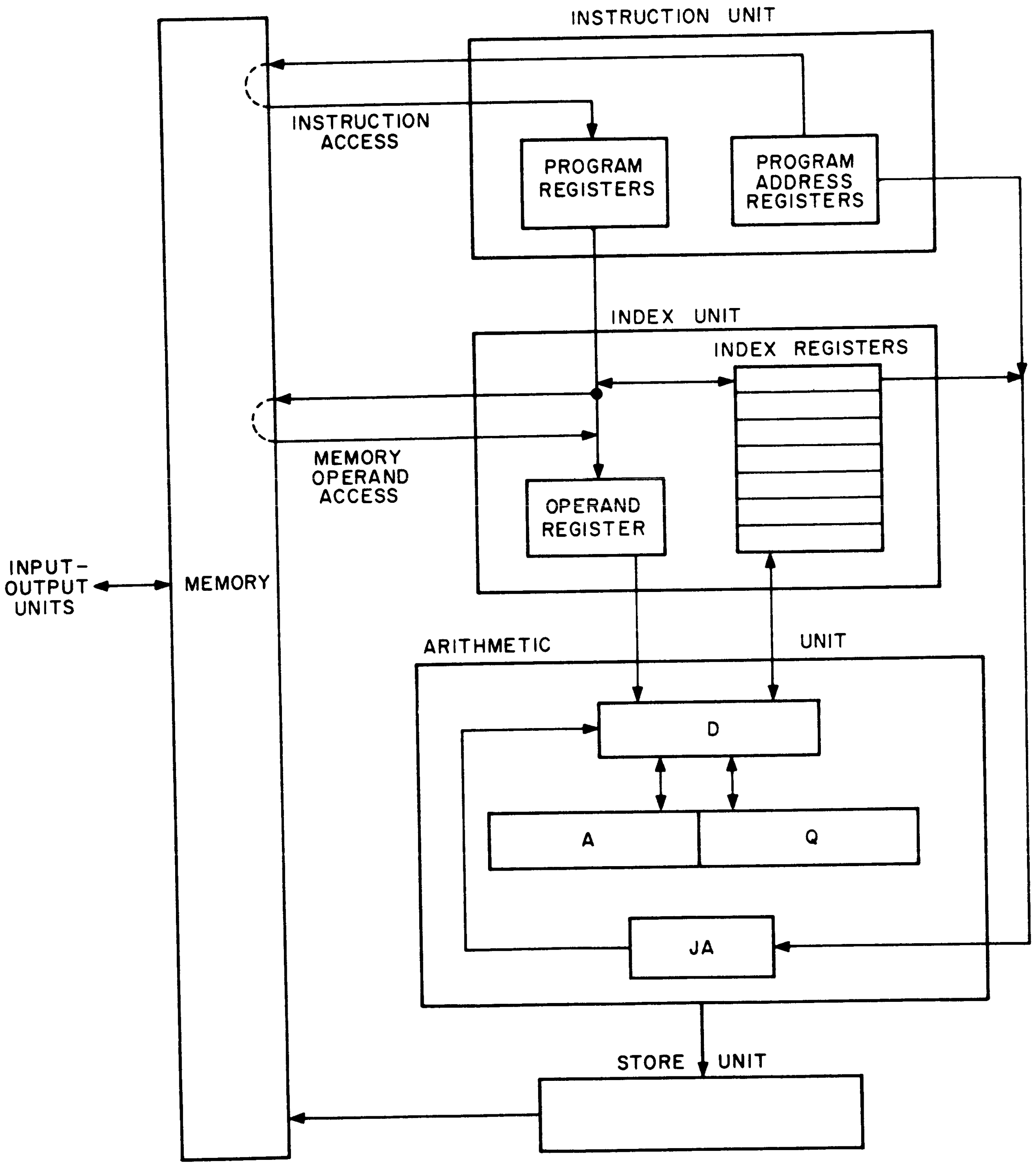


Figure 2. Philco 212 Central Processor

The following table shows the effect of various settings of the C- and Y-bits.

Settings		Effective Address of Instruction Referencing Index Register X	Contents of Index Register After Instruction	C-Bits and Y-Bits Can Be Set By:
C	Y			
0	0	Contents of X + V-field of instruction	Unchanged	TCXZ
1	0	Contents of X + V-field of instruction	Previous contents +1	TCXS
0	1	Contents of Index Register	Previous contents +V-field of instruction	TYXZ
1	1	Contents of Index Register	Previous contents -V-field of instruction	TYXS

Index Register Instructions and Indexable Instructions

Instructions which alter the contents of an index register, set the C- and Y-bits of an index register to one or zero, or transfer information between an index register and the D Register are termed "Index Register Instructions" (see page 63). Instructions which may use the contents of an index register to form an effective address are termed "Indexable Instructions". (Index Register, RPT, DR, SKC and SKF instructions cannot use an index register to form an effective address.)

ARITHMETIC UNIT

The Arithmetic Unit receives instructions and operands from the Index Unit, executes the instructions, and transfers results to the Store Unit, which stores them in memory while the Arithmetic Unit is processing the next instruction. There are four major addressable registers in the Arithmetic Unit: the Data Register (D), the Accumulator (A), the Quotient Register (Q) and the Jump Address Register (JA).

D Register

The D Register is a 48-bit addressable register which can receive information from memory, the A, Q, and JA Registers, and the index registers and their associated C- and Y-bits. The D Register can transfer information to the A, Q, and index registers and to the C- and Y-bits associated with the index registers.

The D Register:

- receives all data transferred between the memory and the Arithemtic Unit.
- receives all data transferred between arithmetic registers.
- contains or transmits the addend in addition, the subtrahend in subtraction, the multiplicand in multiplication, and the divisor in division.
- contains one of two quantities or words being compared.

A Register

The A Register, an addressable 48-bit register, can receive information from memory and the D and Q registers. Information can also be transferred from A to memory and to the D and Q registers.

The A Register:

- contains the augend prior to addition, the minuend prior to subtraction, and the dividend or the more significant half of the dividend in division.
- contains the sum after an addition, the difference after a subtraction, the product or more significant half of the product after a multiplication, and the remainder after a division.
- may contain one of two quantities or words being compared.

Q Register

The Q Register is a 48-bit addressable register which can receive information from memory, the A Register, the D Register, and the Tape Controller. The Q Register can also transfer information to memory and the A and D Registers.

The Q Register:

- contains the multiplier prior to multiplication, the less significant half of a double length product after a multiplication, the less significant half of a double length dividend prior to division, and the quotient after a division.
- may contain one of two quantities or words being compared.
- contains a ''mask'' during an extract operation.
- receives status and fault indications from some input-output devices.

JA Register

The JA Register is an addressable memory-sized register which can receive information from the index registers or from a TIJ instruction (see page 51). The contents of JA can be transferred to the D Register and to a specified memory location by a TJM instruction (see page 50).

The purpose of the JA Register is to record the location of the instruction following a Jump Instruction (see page 53). Thus, if the transfer of control is made from one point in a program to another, a return to the point immediately following the transfer of control is possible by referencing the JA Register. Every Jump Instruction, except JL and JR, automatically causes the location of the next instruction to be placed in the JA Register prior to execution of the instruction. The F-bit of JA is set to zero or one to indicate whether the instruction that follows the Jump Instruction is in the left or right half of a word.

If an exponent fault should occur during floating-point arithmetic (see Chapter 5), the Exponent Fault neon is lighted; a transfer of control to memory location 00000 is made and the address of the next instruction word is placed in JA. The F-bit of JA is set to zero if the fault occurred in a left-half instruction, or to one if the fault occurred in a right-half instruction.

STORE UNIT

When the results of an operation in the Arithmetic Unit are to be stored in memory, the information and its address are transferred to the Store Unit, which then stores the result into memory while the next arithmetic instruction is being executed. Thus, the time required to store results is usually overlapped by the time required to execute the following instruction.

Chapter 4

FIXED–POINT ARITHMETIC

During the execution of any fixed-point arithmetic operation, the computer regards the numbers involved as fixed-point binary numbers (see page 13). Values are significant to 14 decimal digits.

SCALING

A binary point, separating the integral part of a fixed-point binary number from its fractional part, may be assumed anywhere within a word or outside of a word by the programmer. This process of representing any desired number by selecting an appropriate binary point is called scaling, and the number of positions between the computer's sign bit and the assumed point is called the scale factor.

OVERFLOW

Overflow may indicate the loss of a significant bit (see page 13). Overflow during a left shift occurs if a bit that moves into the sign bit is different from the bit that moves out of the sign bit. During arithmetic operations, overflow occurs if the carry into the sign bit of the result is different from the carry out of the sign bit, i.e., the result exceeds the range of the machine's fixed-point numbers. Overflow, as it may occur during an arithmetic operation, is included in the discussion of each operation below.

Overflow Indicator

The Overflow Indicator has two states or conditions, one and zero. Normally, the Indicator is automatically cleared to zero before each Add, Subtract, Multiply, Divide Instruction (including floating-point), Shift Instruction (see page 59), or four of the Index Register Instructions (AIXOL, AIXOR, SIXOL, SIXOR) and during the execution of a JNO or JOF instruction. It is automatically set to one each time overflow occurs.

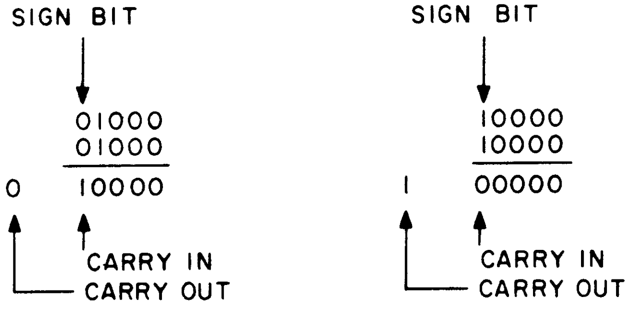
The clearing of the Indicator by the Add, Subtract, Multiply, Divide, Shift, and Index Register Instructions may be inhibited by the ICOS instruction (see page 81) until a convenient time for testing occurs. This inhibition may be removed only by the ICOZ instruction (see page 81). Clearing of the Indicator by the JNO or JOF instructions is never inhibited.

ADDITION AND SUBTRACTION

Addition and subtraction may be thought of as being made bit-by-bit, with carries as necessary into adjacent bits to the left. The following are examples of addition assuming five-bit registers. (The bits are numbered 0-4, and the numbers have a scale factor of four.)

```
     SIGN BIT             SIGN BIT
        ↓                    ↓
 +5   00101           -5   11011
 +4   00100           -4   11100
 ----------           ----------
 +9   01001           -9   10111
```

Examples of overflow, assuming five bit registers:



MULTIPLICATION

In multiplication, the scale factor of the product is equal to the sum of the scale factors of the multiplier and the multiplicand.

Either unrounded or rounded products can be formed as designated by the particular instruction. If a multiplication instruction indicates an unrounded product, a 94-bit product is formed in bits 1-47 of the A and Q Registers. The sign bits of A and Q are the same. If the multiplication instruction indicates a rounded product, a 47-bit product is formed in the A Register. It is the value nearest the product that would have been formed if the multiplication instruction did not indicate a rounded product.

Overflow may occur during multiplication only if two words, each with a sign bit of one which is followed by zeros, are multiplied together. Addition or subtraction in an MAD or MSU instruction (see page 46) may correct the overflow.

DIVISION

In division, the scale factor of the quotient is equal to the difference of the scale factors of the dividend and divisor.

The dividend may be in the A and Q Registers or only in the A Register. If the dividend is in both A and Q, the sign bit of Q is ignored.

If the absolute value of the bits (disregarding an assumed binary point) in a dividend is greater than the absolute value of the bits in the divisor, or if the absolute value of the bits in the dividend is equal to the absolute value of the bits in the divisor and the sign bits of the dividend and the divisor are alike, overflow would occur. However, potential overflow is detected before division and the division is not performed. Instead, the A and Q Registers are altered as indicated in the table that follows, the Overflow Indicator is set to one, and the next instruction is executed.

Register	Divide Instruction	
	DA, DAS	DAQ, DAQS
A	Shifted numerically * one place right into Q.	Shifted numerically* one place right into Q.
Q	Bit 0 contains the sign bit of A and bit 1 contains the bit shifted from A. The rest of the bits are cleared to zero.	Shifted numerically* one place right.

* See SRAQN on page 60

If the store option is used (i.e., the result of an operation is transferred to memory by the instruction that performed the operation) and potential overflow is detected, no store takes place and the specified memory location retains the original divisor.

If potential overflow is not detected and division is performed, the absolute value of the remainder is less than the absolute value of the dividend. The sign of the remainder is the same as the sign of the dividend.

Chapter 5

FLOATING–POINT ARITHMETIC

With floating-point arithmetic the programmer is relieved of the necessity of scaling, and a greater range of values can be expressed in computer words. Values are significant to ten decimal places. For a description of a floating-point word refer to page 14. The floating-point instructions described in Chapter 6 have an F as the first letter of their mnemonic codes.

Double precision floating-point operations are described in Chapter 7.

NORMALIZATION

A floating-point number is in normalized form if the most significant bit of the mantissa immediately follows the first bit. Thus, the value of the sign bit of the mantissa and of the adjacent bit in a normalized floating-point value are different.

Although original operands need not be in normalized form, the computer will always attempt to normalize the result of a floating-point arithmetic operation. The Philco 212 will also normalize the operands of a division before attempting the operation. Normalizing permits the maximum number of significant digits in arithmetic results.

The Philco 212 first examines the mantissa of a result *or of a division operand*. If the mantissa is zero, the number is set to floating-point zero. If the mantissa is not zero and is not normalized, the bits of the mantissa are numerically shifted left until a normalized value results. The exponent is then decremented by the number of shifts which have taken place. If exponent underflow (see next page) occurs during normalization, the number being normalized is set to floating-point zero.

MANTISSA OVERFLOW

Mantissa overflow occurs if the carry into the sign bit of the mantissa is different from the carry out of the sign bit. The Overflow Indicator is *not* set. Instead, mantissa overflow is automatically corrected as indicated below, except in the case of division. In division, potential overflow is detected and the division proceeds as described below.

When mantissa overflow occurs during an addition, subtraction, or multiplication, the mantissa of the result is shifted right one place (decreasing it by a power of 2) and its exponent is increased by one (increasing it by a power of 2).

Before floating-point division takes place, the normalized mantissas of the operands are tested to see if division would produce overflow. If overflow would occur, the dividend is shifted right one place, its exponent is increased by one, and the division is performed.

EXPONENT OVERFLOW AND UNDERFLOW

In the Arithmetic Unit, exponent overflow occurs whenever the carry into the exponent sign bit is one and the carry out is zero, i.e., when an attempt is made to produce a floating-point number which would have an exponent greater than +2047, the largest possible exponent.

Exponent overflow may occur during multiplication or division or during a correction cycle for mantissa overflow. If it occurs during a multiplication or division operation, the overflow may be corrected while the result is being normalized. If exponent overflow still exists after normalization or if it had occurred during a mantissa correction cycle, the Exponent Fault Neon is lighted; a transfer of control to memory location 00000 is effected, and the address of the next instruction word is placed in the JA Register (see page 22).

Exponent underflow occurs whenever the exponent sign bit is zero and the carry out is one, i.e., when an attempt is made to produce an exponent smaller than -2048, the smallest possible exponent.

Exponent underflow may occur during multiplication or division or during any normalization cycle. If it occurs during a normalization cycle, the underflow may correct a previous exponent overflow. If there had been no previous exponent overflow and underflow occurs during a normalization cycle, the result is set to floating-point zero.

ADDITION AND SUBTRACTION

For floating-point addition and subtraction, the Arithmetic Unit arranges the floating-point word with the smaller exponent so that its exponent is effectively equal to the larger exponent in the other word. This is done by shifting right the mantissa of the value with the smaller exponent the number of places equal to the difference between the exponents if the absolute value of the difference is less than 36 (or less than 71 when in double precision mode). If the value with the smaller exponent is the addend or subtrahend in the D Register, its mantissa is shifted in the D Register. *(The exponent of the addend or subtrahend in the D Register is not changed from its original value.)* If the absolute value of the difference between the exponents is greater than 35 (or greater than 70 when in double precision mode), the operand with the larger exponent becomes the sum, and no time is taken up for shifting.

MULTIPLICATION

When multiplying two floating-point numbers, the Arithmetic Unit adds the exponents and multiplies the mantissas.

Either unrounded or rounded products can be formed. If a product is unrounded, a 70-bit mantissa is formed in bits 1-35 of the A and Q Registers. The exponents (bits 36-47) and sign bits in A and Q are alike. If a product is rounded, a 35-bit mantissa is formed in the A Register. It is the value nearest the product that would have been formed, prior to normalization, if the multiplication had not been rounded.

DIVISION

In floating-point division, the Arithmetic Unit subtracts the exponent of the divisor from the exponent of the dividend and divides the mantissa of the dividend by the mantissa of the divisor.

The dividend may be in the A and Q Registers or only in the A Register. If the dividend is in both A and Q, the mantissa is 70 bits and is in bits 1-35 of A and of Q. The exponent (bits 36-47) and the sign bit of the dividend used are in the A Register; the exponent and the sign bit in the Q Register are ignored.

The exponent of the remainder is 35 less than the exponent of the original normalized dividend. If the exponent of the remainder goes out of range, the remainder is set to floating-point zero.

Division by zero is detected during the normalization process (see page 30). If a divisor with a mantissa equal to zero is detected, division does not proceed. However, the dividend is normalized. The Exponent Fault Neon is lighted; a transfer of control to memory location 00000 is made and the address of the next instruction word is placed in the JA Register (see page 22).

If a dividend with a zero mantissa is detected, and the mantissa of the divisor is not zero, the quotient and remainder are floating-point zero.

Chapter 6

PHILCO 212 INSTRUCTIONS

The mnemonic* codes for all Philco 212 instructions are given below together with a description of their functions. (A mnemonic code with F as its first letter specifies floating-point arithmetic.) The quaternary† representation of the eight-bit machine language command appears with each mnemonic. These codes represent the command field of the instruction only; the entire instruction also consists of an address field (see page 16). When an instruction calls for an operand in memory, the memory location is specified in the address field. When the address field specifies something other than an address, it is so noted in the description.

If the Central Processor attempts to decode a command field and finds it is not one of those listed herein, a command fault is indicated on the Operator's Console and the Central Processor halts.

No bits in any addressable register are altered by an instruction, except those specified by the description of that instruction. (Appendix A defines the symbols in the logic equations and flow charts that are used to define the functions of some instructions.)

ADD INSTRUCTIONS

Add Instructions add the contents (or the absolute value of the contents) of a register or a specified memory location to the contents of the A Register. Clear and Store options are available.

All Add instructions first clear the Overflow Indicator to zero unless an ICOS instruction (see page 81) has been given and is still in effect.

* The term "mnemonic code" refers to the code which the programmer writes for an instruction. The mnemonic code is converted into machine language by the Translator-Assembler-Compiler, TAC (see Philco 2000 TAC Manual, TM-11).

† Quaternary code uses a single digit to represent two bits, as follows:

0 = 00	2 = 10
1 = 01	3 = 11

AM 1000 *Add Memory*

FAM 3000

$$(M) \longrightarrow D$$
$$(A) + (M) \longrightarrow A$$

The AM/FAM instruction transfers the operand from the specified memory location to the D register, then adds the operand to the contents of the A Register. The sum replaces the contents of A and the operand remains in D. For FAM, the operand in D may have been arranged to make the exponents of A and D equal for addition (see page 30).

AMS 1001 *Add Memory and Store*

FAMS 3001

$$(A) + (M) \longrightarrow A, \text{ D and M}$$

The AMS/FAMS instruction performs as an AM/FAM instruction, then transfers the sum from the A Register to the D Register and to the original memory location. The sum replaces the contents of A, D and the original memory location.

CAM 1002 *Clear and Add Memory*

FCAM 3002

$$(M) \longrightarrow \text{D and A}$$

The CAM/FCAM instruction transfers the operand from the specified memory location to the D Register and to the A Register. The operand from memory replaces the contents of A and D. For FCAM, the final contents of A are normalized.

CAMS 1003 *Clear, Add Memory and Store*

FCAMS 3003

$$(M) \longrightarrow A, \text{ D and M}$$

The CAMS/FCAMS instruction performs as a CAM/FCAM instruction, then transfers the final contents of the A Register to the D Register and to the original memory location. The final contents of A remain in A, D, and the original memory location.

AMA 1010 *Add Memory Absolute*

FAMA 3010

$$(M) \longrightarrow D$$
$$(A) + |(M)| \longrightarrow A$$

The AMA/FAMA instruction transfers the operand from the specified memory location to the D Register, then adds the absolute value of the operand to the contents of the A Register. The sum replaces A and the operand from memory remains in D. For FAMA, the operand in D may have been arranged for addition (see page 30).

AMAS 1011 *Add Memory Absolute and Store*
FAMAS 3011

$$(A) + |(M)| \longrightarrow A \;\; D \text{ and } M$$

The AMAS/FAMAS instruction performs as an AMA/FAMA instruction, then transfers the sum from the A Register to the D Register, and to the original memory location. The sum replaces the contents of A, D, and the original memory location.

CAMA 1012 *Clear and Add Memory Absolute*
FCAMA 3012

$$(M) \longrightarrow D$$
$$|(M)| \longrightarrow A$$

The CAMA/FCAMA instruction transfers the operand from the specified memory location to the D Register, and the absolute value of the operand to the A Register. The absolute value of the operand from memory replaces the contents of A and the operand from memory remains in D. For FCAMA, the final contents of A are normalized.

CAMAS 1013 *Clear, Add Memory Absolute and Store*
FCAMAS 3013

$$|(M)| \longrightarrow A, D \text{ and } M$$

The CAMAS/FCAMAS instruction performs as a CAMA/FCAMA instruction, then transfers the final contents of the A Register to the D Register, and to the original memory location. The final contents of the A Register remain in A, D, and the original memory location.

AQ 1020 *Add Q*
FAQ 3020

$$(Q) \longrightarrow D$$
$$(A) + (Q) \longrightarrow A$$

The AQ/FAQ instruction transfers the operand in the Q Register to the D Register, then adds the operand to the contents of the A Register. The sum replaces the contents of A, and the operand from Q replaces the contents of D. For FAQ, the final contents of D may have been arranged for addition (see page 30).

AQS 1021 *Add Q and Store*
FAQS 3021

$$(A) + (Q) \longrightarrow A, D \text{ and } M$$

The AQS/FAQS instruction performs as an AQ/FAQ instruction, then transfers the sum to the D Register and to the original memory location. The sum replaces the contents of A, D, and the original memory location.

CAQ **1022** *Clear and Add Q*
FCAQ **3022**

$$(Q) \longrightarrow D \text{ and } A$$

The CAQ/FACQ instruction transfers the operand in the Q Register to the D Register and to the A Register. The operand from Q replaces the contents of A and D. For FCAQ, the final contents of A are normalized.

CAQS **1023** *Clear, Add Q and Store*
FCAQS **3023**

$$(Q) \longrightarrow A, D \text{ and } M$$

The CAQS/FCAQS instruction performs as a CAQ/FCAQ instruction, then transfers the final contents of the A Register to the D Register and to the original memory location. The final contents of A remain in A, D, and the original memory location.

AQA **1030** *Add Q Absolute*
FAQA **3030**

$$(Q) \longrightarrow D$$
$$(A) + |(Q)| \longrightarrow A$$

The AQA/FAQA instruction transfers the operand from the Q Register to the D Register, then adds the absolute value of the operand to the contents of the A Register. The sum replaces the contents of A and the operand from Q remains in D. For FAQA, the operand in D may have been arranged for addition (see page 30).

AQAS **1031** *Add Q Absolute and Store*
FAQAS **3031**

$$(A) + |(Q)| \longrightarrow A, D \text{ and } M$$

The AQAS/FAQAS instruction performs as an AQA/FAQA instruction, then transfers the sum in the A Register to the D Register and to the original memory location. The sum replaces the contents of A, D, and the original memory location.

CAQA **1032** *Clear and Add Q Absolute*
FCAQA **3032**

$$(Q) \longrightarrow D$$
$$|(Q)| \longrightarrow A$$

The CAQA/FCAQA instruction transfers the operand from the Q Register to the D Register, and the absolute value of the operand to the A Register. The operand from Q remains in D and the absolute value of the operand is in A. For FCAQA, the final contents of A are normalized.

CAQAS 1033 *Clear, Add Q Absolute and Store*
FCAQAS 3033

$$|(Q)| \longrightarrow \text{A, D and M}$$

The CAQAS/FCAQAS instruction performs as a CAQA/FCAQA instruction, then transfers the final contents of the A Register to the D Register and to the original memory location. The final contents of the A Register remain in A, D, and the original memory location.

AD 1330 *Add D*
FAD 3330

$$(A) + (D) \longrightarrow A$$

The AD/FAD instruction adds the contents of the D Register to the contents of the A Register. The sum replaces the contents of A. For FAD, the original contents of D may have been arranged for addition (see page 30).

SUBTRACT INSTRUCTIONS

Subtract instructions subtract the contents (or the absolute value of the contents) of a register or a specified memory location from the contents of the A Register. Clear and store options are available.

All Subtract Instructions first clear the Overflow Indicator to zero unless an ICOS instruction has been given and is still in effect.

SM 1100 *Subtract Memory*
FSM 3100

$$(M) \longrightarrow D$$
$$(A) - (M) \longrightarrow A$$

The SM/FSM instruction transfers the operand from the specified memory location to the D Register, then subtracts the operand from the contents of the A Register. The difference replaces the contents of the A Register and the operand from memory remains in D. For FSM, the operand in D may have been arranged for subtraction (see page 30).

SMS 1101 *Subtract Memory and Store*
FSMS 3101

$$(A) - (M) \longrightarrow \text{A, D and M}$$

The SMS/FSMS instruction performs as an SM/FSM instruction, then transfers the difference from the A Register to the D Register and to the original memory location. The difference replaces the contents of A, D and the original memory location.

CSM 1102 *Clear and Subtract Memory*
FCSM 3102

$$(M) \rightarrow D$$
$$-(M) \rightarrow A$$

The CSM/FCSM instruction transfers the operand from the specified memory location to the D Register and the two's complement of the operand to the A Register. The operand from memory remains in D and the two's complement of the operand replaces the contents of A. For FCSM, the final contents of A are normalized.

CSMS 1103 *Clear, Subtract Memory and Store*
FCSMS

$$-(M) \rightarrow A,\ D \text{ and } M$$

The CSMS/FCSMS instruction performs as a CSM/FCSM instruction, then transfers the final contents of the A Register to the D Register and to the original memory location. The final contents of A remain in A, D and the original memory location.

SMA 1110 *Subtract Memory Absolute*
FSMA 3110

$$(M) \rightarrow D$$
$$(A) - |(M)| \rightarrow A$$

The SMA/FSMA instruction transfers the operand from the specified memory location to the D Register, then subtracts the absolute value of the operand from the contents of the A Register. The difference replaces the contents of the A Register and the operand from memory remains in D. For FSMA, the operand in D may have been arranged for subtraction (see page 30).

SMAS 1111 *Subtract Memory Absolute and Store*
FSMAS 3111

$$(A) - |(M)| \rightarrow A,\ D \text{ and } M$$

The SMAS/FSMAS instruction performs as an SMA/FSMA instruction, then transfers the difference from the A Register to the D Register and to the original memory location. The difference replaces the contents of A, D and the original memory location.

CSMA 1112 *Clear, Subtract Memory Absolute*
FCSMA 3112

$$(M) \rightarrow D$$
$$-|(M)| \rightarrow A$$

The CSMA/FCSMA instruction transfers the operand from the specified memory location to the D Register, and the two's complement of the absolute value of the operand to the A Register. The two's complement of the absolute value of the operand from the specified memory location replaces the contents of A and the operand from memory remains in D. For FCSMA, the final contents of A are normalized.

CSMAS 1113 *Clear, Subtract Memory Absolute and Store*
FCSMAS 3113

$$-|(M)| \longrightarrow \text{A, D and M}$$

The CSMAS/FCSMAS instruction performs as a CSMA/FCSMA instruction, then transfers the final contents of the A Register to the D Register and to the original memory location. The final contents of the A Register remain in A, D, and the original memory location.

SQ 1120 *Subtract Q*
FSQ 3120

$$(Q) \longrightarrow D$$
$$(A) - (Q) \longrightarrow A$$

The SQ/FSQ instruction transfers the operand in the Q Register to the D Register and subtracts the operand from the contents of the A Register. The difference replaces the contents of A and the operand from Q remains in D. For FSQ, the operand in D may have been arranged for subtraction (see page 30).

SQS 1121 *Subtract Q and Store*
FSQS 3121

$$(A) - (Q) \longrightarrow \text{A, D and M}$$

The SQS/FSQS instruction performs as an SQ/FSQ instruction, then transfers the difference from the A Register to the D Register and to the original memory location. The difference replaces the contents of A, D, and the original memory location.

CSQ 1122 *Clear and Subtract Q*
FCSQ 3122

$$(Q) \longrightarrow D$$
$$-(Q) \longrightarrow A$$

The CSQ/FCSQ instruction transfers the operand in the Q Register to the D Register, and the two's complement of the operand to the A Register. The operand from the Q Register replaces the contents of D and the two's complement of the operand replaces the contents of A. For FCSQ, the final contents of A are normalized.

CSQS 1123 *Clear, Subtract Q and Store*
FCSQS 3123

$$-(Q) \longrightarrow \text{A, D and M}$$

The CSQS/FCSQS instruction performs as a CSQ/FCSQ instruction, then transfers the final contents of the A Register to the D Register and to the original memory location. The final contents of the A Register remain in A, D, and the original memory location.

SQA 1130 *Subtract Q Absolute*
FSQA 3130

$$(Q) \rightarrow D$$
$$(A) - |(Q)| \rightarrow A$$

The SQA/FSQA instruction transfers the operand from the Q Register to the D Register, then subtracts the absolute value of the operand from the contents of the A Register. The difference replaces the contents of A and the operand from Q remains in D. For FSQA, the operand in D may have been arranged for subtraction (see page 30).

SQAS 1131 *Subtract Q Absolute and Store*
FSQAS 3131

$$(A) - |(Q)| \rightarrow A\text{, } D \text{ and } M$$

The SQAS/FSQAS instruction performs as an SQA/FSQA instruction, then transfers the difference from the A Register to the D Register and to the original memory location. The difference replaces the contents of A, D, and the original memory location.

CSQA 1132 *Clear and Subtract Q Absolute*
FCSQA 3132

$$(Q) \rightarrow D$$
$$-|(Q)| \rightarrow A$$

The CSQA/FCSQA instruction transfers the operand from the Q Register to the D Register, and the two's complement of the absolute value of the operand to the A Register. The two's complement of the absolute value of the operand from Q remains in the A Register and the operand from Q remains in D. For FCSQA, the final contents of A are normalized.

CSQAS 1133 *Clear, Subtract Q Absolute and Store*
FCSQAS 3133

$$-|(Q)| \rightarrow A\text{, } D \text{ and } M$$

The CSQAS/FCSQAS instruction performs as a CSQA/FCSQA instruction, then transfers the final contents of the A Register to the D Register and to the original memory location. The final contents of the A Register remain in A, D, and the original memory location.

SD 1331 *Subtract D*
FSD 3331

$$(A) - (D) \rightarrow A$$

The SD/FSD instruction subtracts the contents of the D Register from the contents of the A Register. The difference replaces the contents of the A Register. For FSD, the original contents of D may have been arranged for subtraction (see page 30).

MULTIPLY INSTRUCTIONS

Multiply Instructions multiply the contents of the Q Register by the contents (or absolute value of the contents) of the A Register or a specified memory location. Round (see page 30) and store options are available.

All Multiply Instructions first clear the Overflow Indicator to zero unless an ICOS instruction has been given and is still in effect.

MM 1200 *Multiply Memory*
FMM 3200

$$(M) \longrightarrow D$$
$$(M) \times (Q) \longrightarrow AQ$$

The MM/FMM instruction transfers the operand (the multiplicand) from the specified memory location to the D Register, then multiplies the operand by the contents of the Q Register (the multiplier).

For fixed-point multiplication, a 94-bit product appears in the A and Q Registers, with the major 47 bits in A and the minor 47 bits in Q. The sign bits of A and Q are the same.

For floating-point multiplication, a 70-bit product of the mantissas appears in bits 1-35 of A and Q. The exponents (bits 36-47) and the sign bits in the A and Q Registers are the same.

The multiplicand for an MM or an FMM instruction remains in D.

MMS 1201 *Multiply Memory and Store*
FMMS 3201

$$(M) \times (Q) \longrightarrow AQ, \text{ then } (A) \longrightarrow D \text{ and } M$$

The MMS/FMMS instruction performs as an MM/FMM instruction, then transfers the major half of the product from the A Register to the D Register and to the original memory location. The major half of the product replaces the contents of D and the original memory location. The minor half of the product remains in Q.

MMR 1202 *Multiply Memory and Round*
FMMR 3202

$$(M) \longrightarrow D$$
$$(M) \times (Q) \longrightarrow A$$

The MMR/FMMR instruction transfers the operand (the multiplicand) from the specified memory location to the D Register, then multiplies the operand by the contents of the Q Register (the multiplier).

For fixed-point multiplication, a 47-bit product, rounded to the value closest to the 94-bit product that would have been formed by an MM instruction, appears in the A Register.

For floating-point multiplication, a 35-bit product of the mantissas appears in bits 1-35 of the A Register. This product is rounded to the value nearest the 70-bit product which would have been formed, prior to normalization, by an FMM instruction. The exponent appears in bits 36-47 of the A Register.

For either MMR or FMMR, the multiplier appears in Q and the multiplicand appears in D.

MMRS 1203 *Multiply Memory, Round and Store*
FMMRS 3203

$$(M) \times (Q) \longrightarrow A, D \text{ and } M$$

The MMRS/FMMRS instruction performs as an MMR/FMMR instruction, then transfers the product from the A Register to the D Register and to the original memory location.

The product remains in A, D and the original memory location. The multiplier appears in Q.

MMA 1210 *Multiply Memory Absolute*
FMMA 3210

$$(M) \longrightarrow D$$
$$|(M)| \times (Q) \longrightarrow AQ$$

The MMA/FMMA instruction transfers the operand from the specified memory location to the D Register, then multiplies the absolute value of the operand (the multiplicand) by the contents of the Q Register (the multiplier).

For fixed-point multiplication, a 94-bit product appears in the A and Q Registers, with the major 47-bits in A and the minor 47 bits in Q. The sign bits of A and Q are the same.

For floating-point multiplication, a 70-bit product of the mantissas of the operands appears in bits 1-35 of A and Q. The exponents (bits 36-47) and the sign bits in the A and Q Registers are the same.

For MMA or FMMA the operand from memory remains in D.

MMAS 1211 *Multiply Memory Absolute and Store*
FMMAS 3211

$$|(M)| \times (Q) \longrightarrow AQ, \text{ then } (A) \longrightarrow D \text{ and } M$$

The MMAS/FMMAS instruction performs as an MMA/FMMA instruction, then transfers the major half of the product from the A Register to the D Register and to the original memory location. The major half of the product remains in A, D, and the original memory location. The minor half of the product remains in Q.

MMAR 1212 *Multiply Memory Absolute and Round*
FMMAR 3212

$$(M) \rightarrow D$$
$$|(M)| \times (Q) \rightarrow A$$

The MMAR/FMMAR instruction transfers the operand from the specified memory location to the D Register, then multiplies the absolute value of the operand (the multiplicand) by the contents of the Q Register (the multiplier).

For fixed-point' multiplication, a 47-bit product, rounded to the value closest to the 94-bit product that would have been formed by an MMA instruction, appears in the A Register.

For floating-point multiplication, a 35-bit product of the mantissas appears in bits 1-35 of the A Register. The product is rounded to the value nearest the 70-bit product that would have been formed, prior to normalization, by an FMMA instruction. The exponent appears in bits 36-47 of the A Register.

For either an MMAR or an FMMAR, the multiplier appears in Q and the multiplicand appears in D.

MMARS 1213 *Multiply Memory Absolute, Round and Store*
FMMARS 3213

$$|(M)| \times (Q) \rightarrow A, D \text{ and } M$$

The MMARS/FMMARS instruction performs as an MMAR/FMMAR instruction, then transfers the product from the A Register to the D Register and to the original memory location.

The product remains in A, D, and the original memory location. The multiplier appears in Q.

MA 1220 *Multiply A*
FMA 3220

$$(A) \rightarrow D$$
$$(A) \times (Q) \rightarrow AQ$$

The MA/FMA instruction transfers the operand (the multiplicand) from the A Register to the D Register, then multiplies the operand by the contents of the Q Register (the multiplier).

For fixed-point multiplication, a 94-bit product appears in the A and Q Registers, with the major 47 bits in A and the minor 47 bits in Q. The sign bits of A and Q are the same.

For floating-point multiplication, a 70-bit product of the mantissas appears in bits 1-35 of A and Q. The exponents (bits 36-47) and the sign bits in the A and Q Registers are the same.

The mulitplicand for an MA or an FMA instruction remains in D.

MULTIPLY

MAS 1221 *Multiply A and Store*
FMAS 3221

$$(A) \times (Q) \longrightarrow AQ, \text{ then } (A) \longrightarrow D \text{ and } M$$

The MAS/FMAS instruction performs as an MA/FMA instruction, then transfers the major half of the product from the A Register to the D Register and to the specified memory location. The major half of the product replaces the contents of D and of the original memory location. The minor half of the product remains in Q.

MAR 1222 *Multiply A and Round*
FMAR 3222

$$(A) \longrightarrow D$$
$$(A) \times (Q) \longrightarrow A$$

The MAR/FMAR instruction transfers the operand (the multiplicand) from the A Register to the D Register, then multiplies the operand by the contents of the Q Register (the multiplier).

For fixed-point multiplication, a 47-bit product, rounded to the value closest to the 94-bit product that would have been formed by an MA instruction, appears in the A Register.

For floating-point multiplication, a 35-bit product of the mantissas appears in bits 1-35 of the A Register. This product is rounded to the value closest to the 70-bit product that would have been formed, prior to normalization, by an FMA instruction. The exponent appears in bits 36-47 of the A Register.

For either MAR or FMAR, the multiplier appears in Q and the multiplicand appears in D.

MARS 1223 *Multiply A, Round and Store*
FMARS 3223

$$(A) \times (Q) \longrightarrow A, D \text{ and } M$$

The MARS/FMARS instruction performs as an MAR/FMAR instruction, then transfers the product from the A Register to the D Register and to the original memory location.

The product remains in A, D, and the original memory location. The multiplier appears in Q.

MAA 1230 *Multiply A Absolute*
FMAA 3230

$$(A) \longrightarrow D$$
$$|(A)| \times (Q) \longrightarrow AQ$$

The MAA/FMAA instruction transfers the operand from the A Register to the D Register, then multiplies the absolute value of the operand in the D Register (the mulitplicand) by the contents of the Q Register (the multiplier).

For fixed-point multiplication, a 94-bit product appears in the A and the Q Registers, with the major 47 bits in A and the minor 47 bits in Q. The sign bits of A and Q are the same.

For floating-point multiplication, a 70-bit product of the mantissas appears in bits 1-35 of A and Q. The exponents (bits 36-47) in the A and Q Registers are the same.

For MAA or FMAA, the operand from memory remains in D.

MAAS 1231 *Multiply A Absolute and Store*
FMAAS 3231

$$|(A)| \times (Q) \longrightarrow AQ, \text{ then } (A) \longrightarrow D \text{ and } M$$

The MAAS/FMAAS instruction performs as an MAA/FMAA instruction, then transfers the major half of the product from the A Register to the D Register and to the original memory location. The major half of the product remains in A, D, and the original memory location.

MAAR 1232 *Multiply A Absolute and Round*
FMAAR 3232

$$(A) \longrightarrow D$$
$$|(A)| \times (Q) \longrightarrow A$$

The MAAR/FMAAR instruction transfers the operand from the A Register to the D Register, then multiplies the absolute value of the operand (the multiplicand) by the contents of the Q Register (the multiplier).

For fixed-point multiplication, a 47-bit product, rounded to the value nearest the 94-bit product that would have been formed by an MAA instruction, appears in the A Register.

For floating-point multiplication, a 35-bit product of the mantissas appears in bits 1-35 of the A Register. This product is rounded to the value nearest the 70-bit product that would have been formed, prior to normalization, by an FMAA instruction. The exponent appears in bits 36-47 of the A Register.

For either an MAAR or an FMAAR, the multiplier appears in Q and the multiplicand appears in D.

MAARS 1233 *Multiply A Absolute, Round and Store*
FMAARS 3222

$$|(A)| \times (Q) \longrightarrow A, D \text{ and } M$$

The MAARS/FMAARS instruction performs as an MAAR/FMAAR instruction, then transfers the product from the A Register to the D Register and to the original memory location.

The product remains in A, D, and the original memory location. The multiplier appears in Q.

MAD **1320** *Multiply and Add*
FMAD **3320**

$$(A)_a \longrightarrow D$$
$$(M) \times (Q) \longrightarrow A$$
$$(A)_z + (A)_a \longrightarrow A$$

The MAD/FMAD instruction transfers the operand (the multiplicand) from the specified memory location to the D Register. The contents of A are then transferred to an unaddressable control register. The contents of the D Register are multiplied by the contents of the Q Register (the multiplier) forming a rounded product in A. The contents of the unaddressable control register are then added to the contents of A. The sum replaces the contents of the A Register. The original contents of A are in D.[†] For an FMAD, the contents of D may have been arranged for addition (see page 30). The multiplier remains in Q.

MSU **1321** *Multiply and Subtract*
FMSU **3321**

$$(A)_a \longrightarrow D$$
$$(M) \times (Q) \longrightarrow A$$
$$(A)_z - (A)_a \longrightarrow A$$

The MSU/FMSU instruction is the same as the MAD/FMAD instruction, except that the original contents of A are subtracted from the rounded product in A. The difference replaces the contents of the A Register and the original contents of A are in D.[†] For an FMSU, the contents of D may have been arranged for subtraction (see page 30).

DIVIDE INSTRUCTIONS

Divide Instructions divide the contents of the A Register or of the A and Q Registers by the contents of a specified memory location. Store options are available.

All Divide Instructions first clear the Overflow Indicator to zero unless an ICOS instruction has been given and is still in effect.

[†] Fixed-point overflow is indicated only after the addition or subtraction have been completed. If overflow occurs during the multiplication and is corrected after the addition or subtraction, no overflow will be indicated.

DAQ 1300 *Divide A and Q*
FDAQ 3300

$$(M) \longrightarrow D$$
$$(AQ) \div M \longrightarrow Q$$
$$\text{Remainder} \longrightarrow A$$

The DAQ/FDAQ instruction transfers the operand from the specified memory location to the D Register and divides the operand into the contents of the A and Q Registers, treating the contents of the A Register as the major half of the dividend and the contents of the Q Register as the minor half. The exponent (if FDAQ) and the sign bit in Q are ignored. The quotient is developed in the Q Register and the remainder appears in the A Register. The operand from the specified memory location remains in D. For FDAQ, the final contents of the D Register are normalized.

DAQS 1301 *Divide A and Q and Store*
FDAQS 3301

$$(AQ) \div (M) \longrightarrow Q,\ D \text{ and } M$$
$$\text{Remainder} \longrightarrow A$$

The DAQS/FDAQS instruction performs as a DAQ/FDAQ instruction, then transfers the quotient in the Q Register to the D Register and to the original memory location. The quotient remains in Q, D and the original memory location. The remainder appears in the A Register.

DA 1302 *Divide A*
FDA 3302

$$(M) \longrightarrow D$$
$$(A) \div (M) \longrightarrow Q$$
$$\text{Remainder} \longrightarrow A$$

The DA/FDA instruction transfers the operand from the specified memory location to the D Register, then divides the operand into the contents of the A Register. The quotient is developed in the Q Register and the remainder in the A Register. The operand from the specified memory location remains in D. For FDA, the final contents of the D Register are normalized.

DAS 1303 *Divide A and Store*
FDAS 3303

$$(A) \div (M) \longrightarrow Q,\ D \text{ and } M$$
$$\text{Remainder} \longrightarrow A$$

The DAS/FDAS instruction performs as a DA/FDA instruction, then transfers the quotient in the Q Register to the D Register and to the original memory location. The quotient remains in Q, D, and the original memory location. The remainder appears in the A Register.

CLEAR INSTRUCTIONS Clear Instructions clear the contents of a register or a specified memory location to 48 zero bits.

CM 0100 *Clear Memory*

$$0 \longrightarrow M_{0\text{-}47}$$

The CM instruction clears the contents of a specified memory location to 48 zero bits.

DIVIDE

CLEAR

CA 0111 *Clear A*

$$0 \rightarrow A_{0-47}$$

The CA instruction clears the contents of the A Register to 48 zero bits.

CQ 0122 *Clear Q*

$$0 \rightarrow Q_{0-47}$$

The CQ instruction clears the contents of the Q Register to 48 zero bits.

CD 0133 *Clear D*

$$0 \rightarrow D_{0-47}$$

The CD instruction clears the contents of the D Register to 48 zero bits.

TRANSFER INSTRUCTIONS

Transfer Instructions transfer information from one register to another, from a specified memory location to a register, or from a register to a specified memory location. The register or memory location from which a transfer is made remains unchanged after the transfer.

TMA 0101 *Transfer Memory to A*

$$(M) \rightarrow D \text{ and } A$$

The TMA instruction transfers the operand from the specified memory location to the D Register and to the A Register.

TMQ 0102 *Transfer Memory to Q*

$$(M) \rightarrow D \text{ and } Q$$

The TMQ instruction transfers the operand from the specified memory location to the D Register and to the Q Register.

TMD 0103 *Transfer Memory to D*

$$(M) \rightarrow D$$

The TMD instruction transfers the operand from the specified memory location to the D Register.

TAM 0110 *Transfer A to Memory*

$$(A) \rightarrow D \text{ and } M$$

The TAM instruction transfers the contents of the A Register to the D Register and to the specified memory location.

TAQ 0112 *Transfer A to Q*

$$(A) \rightarrow D \text{ and } Q$$

The TAQ instruction transfers the contents of the A Register to the D Register and to the Q Register.

TAD 0113 *Transfer A to D*

$$(A) \longrightarrow D$$

The TAD instruction transfers the contents of the A Register to the D Register.

TQM 0120 *Transfer Q to Memory*

$$(Q) \longrightarrow D \text{ and } M$$

The TQM instruction transfers the contents of the Q Register to the D Register and to the specified memory location.

TQA 0121 *Transfer Q to A*

$$(Q) \longrightarrow D \text{ and } A$$

The TQA instruction transfers the contents of the Q Register to the D Register and to the A Register.

TQD 0123 *Transfer Q to D*

$$(Q) \longrightarrow D$$

The TQD instruction transfers the contents of the Q Register to the D Register.

TDM 0130 *Transfer D to Memory*

$$(D) \longrightarrow M$$

The TDM instruction transfers the contents of the D Register to the specified memory location.

TDA 0131 *Transfer D to A*

$$(D) \longrightarrow A$$

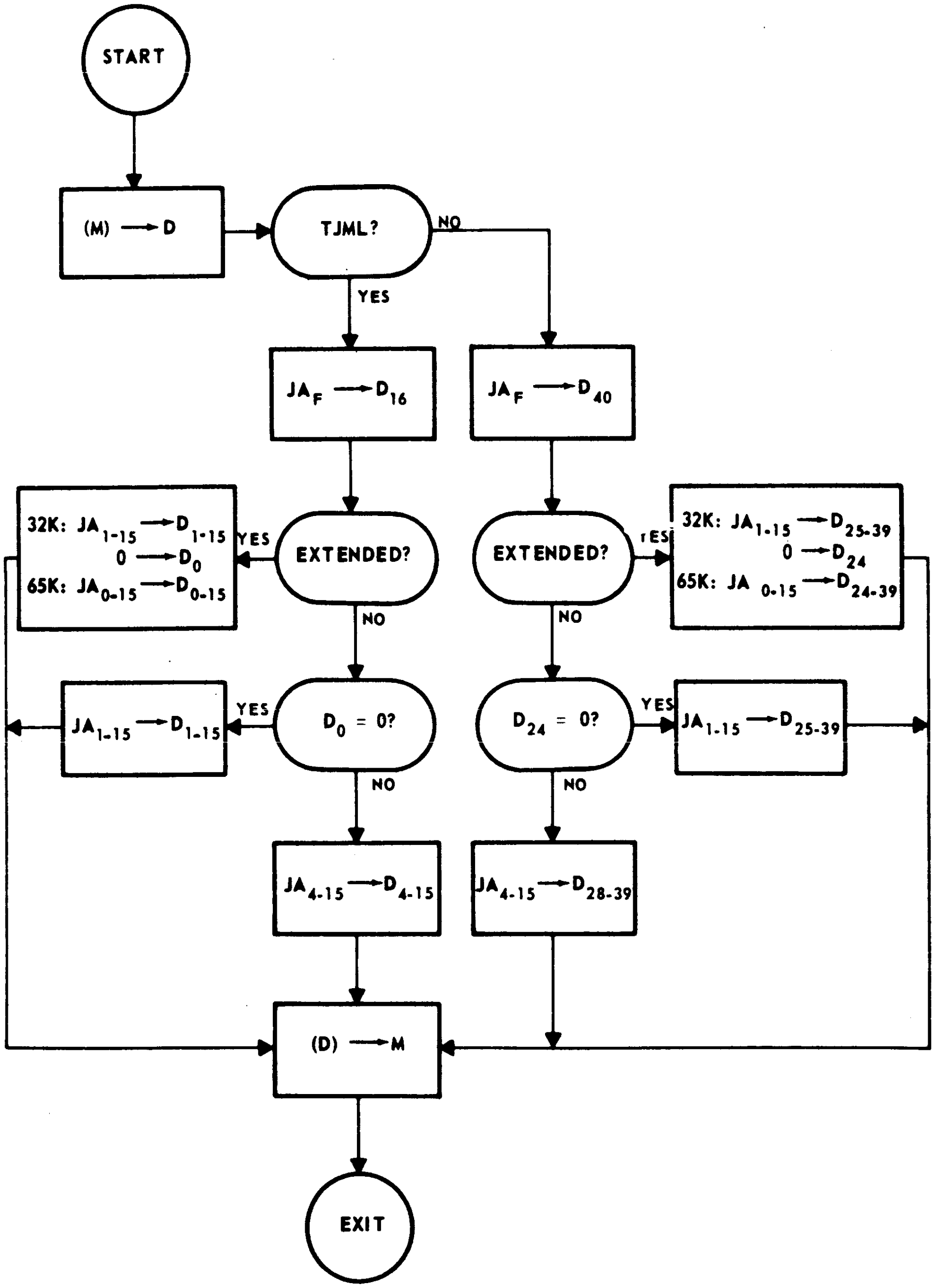
The TDA instruction transfers the contents of the D Register to the A Register.

TDQ 0132 *Transfer D to Q*

$$(D) \longrightarrow Q$$

The TDQ instruction transfers the contents of the D Register to the Q Register.

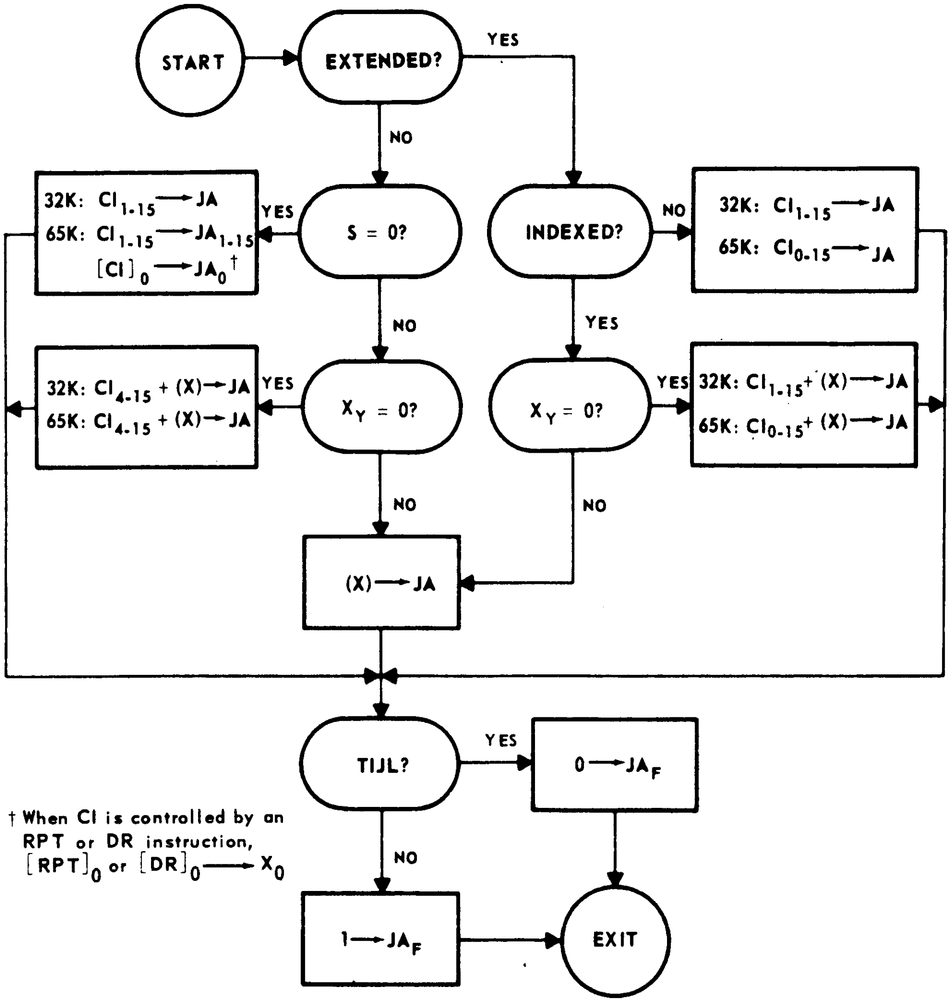
TJML **0020** *Transfer the Contents of the JA Register to Memory*
TJMR **2020**


START
(M) → D
TJML?
NO
YES
$JA_F \rightarrow D_{16}$
$JA_F \rightarrow D_{40}$
EXTENDED?
YES
32K: $JA_{1-15} \rightarrow D_{1-15}$
$0 \rightarrow D_0$
65K: $JA_{0-15} \rightarrow D_{0-15}$
NO
EXTENDED?
YES
32K: $JA_{1-15} \rightarrow D_{25-39}$
$0 \rightarrow D_{24}$
65K: $JA_{0-15} \rightarrow D_{24-39}$
NO
$D_0 = 0?$
YES
$JA_{1-15} \rightarrow D_{1-15}$
NO
$D_{24} = 0?$
YES
$JA_{1-15} \rightarrow D_{25-39}$
NO
$JA_{4-15} \rightarrow D_{4-15}$
$JA_{4-15} \rightarrow D_{28-39}$
(D) → M
EXIT

The TJM instruction transfers the operand from the specified memory location to the D Register, then transfers either 12 or 15 bits (depending on the setting of the S-bit of the specified address field in D) and the F-bit from the Jump Address Register to the specified address field (TJML or TJMR for left or right) of the D Register, unless this instruction is extended by an EXT instruction (see pages 88-92). Only the specified address part of the word in D is altered. The contents of the D Register are then transferred to the specified memory location. The altered operand remains in D.

TIJL 0022 *Transfer the Instruction Address Field to JA*

TIJR 2022


START
EXTENDED?
YES
NO
32K: $CI_{1-15} \rightarrow JA$
65K: $CI_{1-15} \rightarrow JA_{1-15}$
$[CI]_0 \rightarrow JA_0$ †
YES
S = 0?
INDEXED?
NO
32K: $CI_{1-15} \rightarrow JA$
65K: $CI_{0-15} \rightarrow JA$
NO
YES
32K: $CI_{4-15} + (X) \rightarrow JA$
65K: $CI_{4-15} + (X) \rightarrow JA$
YES
$X_Y = 0?$
$X_Y = 0?$
YES
32K: $CI_{1-15} + (X) \rightarrow JA$
65K: $CI_{0-15} + (X) \rightarrow JA$
NO
NO
$(X) \rightarrow JA$
TIJL?
YES
$0 \rightarrow JA_F$
NO
$1 \rightarrow JA_F$
EXIT

† When CI is controlled by an RPT or DR instruction, $[RPT]_0$ or $[DR]_0 \rightarrow X_0$

The TIJ instruction places its effective address in the JA Register and sets the F-bit of the JA Register to zero (if TIJL) or to one (if TIJR).

On a 65K system, if this instruction is not indexed and is not extended by an EXT instruction (see pages 88-92), the first bit of the JA Register is set equal to the first bit of the 16-bit address of the memory location in which the instruction is stored unless this instruction is controlled by an RPT or DR instruction. If this instruction is controlled by an RPT or DR instruction and is not indexed or extended, the first bit of the JA Register is set equal to the first bit of the 16-bit address of the memory location in which the RPT or DR instruction is stored.

TTD 2010 *Transfer from Toggle Register to D*

(Toggle Register) → D

The TTD instruction transfers the word established in a manually operated Toggle Register to D. The Toggle Register is a 48-bit register composed of 48 switches on the control console. Each switch may be placed in the on or off position to correspond to a binary one or zero, respectively.

TCM **0011** *Transfer from Console Typewriter to Memory*

$$CT \longrightarrow D_{42\text{-}47}$$

$$(D)_Z \longrightarrow M$$

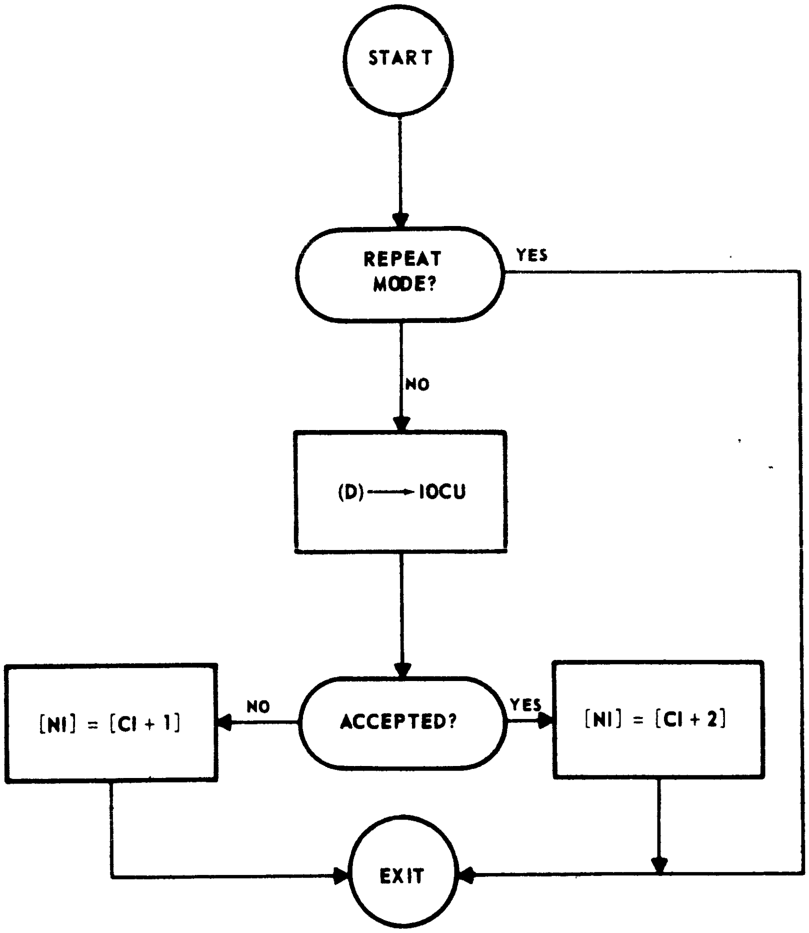
The TCM instruction transfers one character from the Console Typewriter keyboard to the six rightmost bit positions of the D Register without altering the remaining positions of D. The entire contents of the D Register are then transferred to a specified memory location.

TDC **2011** *Transfer from D to Console Typewriter*

$$D_{0\text{-}5} \longrightarrow CT$$

The TDC instruction transfers the character in the six leftmost bit positions of the D Register to the Console Typewriter. The character is then typed.

TIO **0010** *Transfer Control to Input-Output*



The TIO instruction transfers the word from the D Register to the appropriate input-output system that will interpret it as an input-output order (see page 17). If the input-output order is acceptable to the input-output system, the instruction following the TIO is skipped and the instruction beyond is executed. If the input-output order is not accepted, control is transferred to the instruction immediately following the TIO.

The address field of the TIO instruction generally indicates the starting address in memory of the input-output transmission. On a 65K system, a TIO instruction that is not indexed and is not extended by an EXT instruction (see pages 88-92) uses the value of the first bit of the 16-bit address of the memory location in which the instruction is stored as the first bit of its starting address.

This instruction acts as an NOP if it is under control of an RPT or a DR instruction (see pages 83 and 85).

JUMP INSTRUCTIONS

Jump Instructions effect a transfer of control from the Jump Instruction to any other instruction. This transfer may be unconditional or be dependent on some condition existing in a register or registers. Every Jump Instruction, except JL and JR, first stores the location of the next instruction in the JA Register (unless the Jump Instruction is in the left half of an instruction word controlled by an RPT or DR instruction) and sets the F-bit of the JA Register to zero or one, depending on whether the next instruction is in the left or right half of that location.

On a 65K system, if a Jump Instruction is not indexed and is not extended by an EXT instruction (see pages 88-92), the first bit of JA is set equal to the first bit in the 16-bit address of the memory location in which the instruction is stored unless the Jump Instruction is controlled by an RPT or DR instruction. If the Jump Instruction is controlled by an RPT or DR instruction and is not indexed or extended, the first bit of the JA Register is set equal to the first bit of the 16-bit address of the memory location in which the RPT or DR instruction is stored.

JMPL **0200** *Jump*

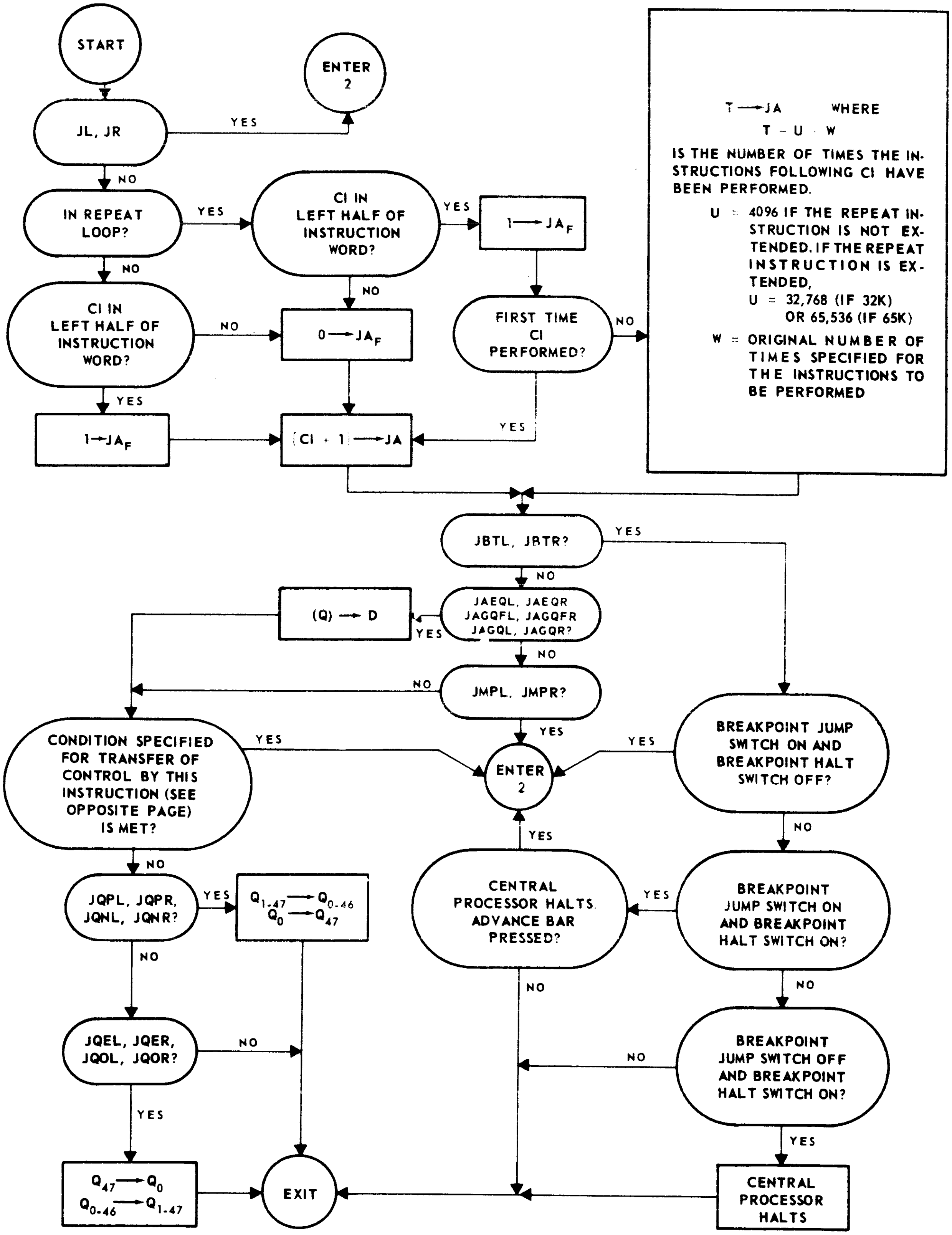
JMPR **2200**

The JMP instruction changes the sequence of instructions by executing the next instruction and any subsequent instructions starting at the location specified in the address field of the JMP instruction. The instruction in the left or right half of the specified location is executed depending on whether the instruction is a JMPL or a JMPR.

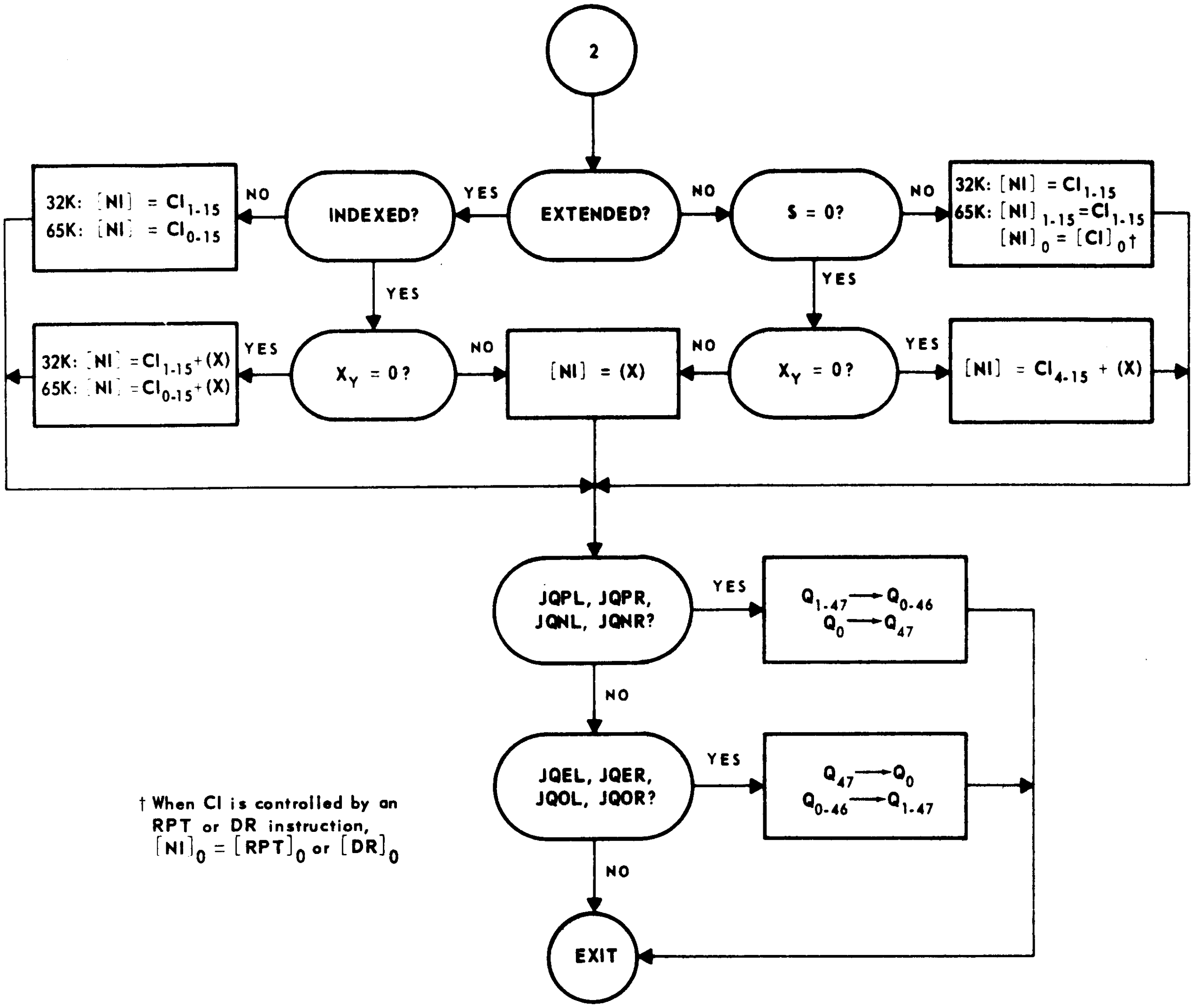
JAZL **0201** *Jump if the Contents of A are Zero*

JAZR **2201**

The JAZ instruction is executed as a JMP instruction *if the contents of the A Register are fixed point zero.* If the contents of the A Register are *not* fixed point zero, the instruction immediately following the JAZ instruction is executed.


START
ENTER 2
JL, JR
YES
NO
IN REPEAT LOOP?
YES
CI IN LEFT HALF OF INSTRUCTION WORD?
YES
1 → JA_F
NO
CI IN LEFT HALF OF INSTRUCTION WORD?
NO
0 → JA_F
FIRST TIME CI PERFORMED?
NO
YES
1 → JA_F
(CI + 1) → JA
T → JA WHERE
T = U - W
IS THE NUMBER OF TIMES THE INSTRUCTIONS FOLLOWING CI HAVE BEEN PERFORMED.
U = 4096 IF THE REPEAT INSTRUCTION IS NOT EXTENDED. IF THE REPEAT INSTRUCTION IS EXTENDED,
U = 32,768 (IF 32K) OR 65,536 (IF 65K)
W = ORIGINAL NUMBER OF TIMES SPECIFIED FOR THE INSTRUCTIONS TO BE PERFORMED
JBTL, JBTR?
YES
NO
JAEQL, JAEQR JAGQFL, JAGQFR JAGQL, JAGQR?
YES
(Q) → D
NO
JMPL, JMPR?
NO
YES
CONDITION SPECIFIED FOR TRANSFER OF CONTROL BY THIS INSTRUCTION (SEE OPPOSITE PAGE) IS MET?
YES
ENTER 2
YES
BREAKPOINT JUMP SWITCH ON AND BREAKPOINT HALT SWITCH OFF?
NO
YES
CENTRAL PROCESSOR HALTS. ADVANCE BAR PRESSED?
YES
BREAKPOINT JUMP SWITCH ON AND BREAKPOINT HALT SWITCH ON?
NO
NO
JQPL, JQPR, JQNL, JQNR?
YES
Q_{1-47} → Q_{0-46}
Q_0 → Q_47
NO
NO
BREAKPOINT JUMP SWITCH OFF AND BREAKPOINT HALT SWITCH ON?
NO
JQEL, JQER, JQOL, JQOR?
NO
YES
YES
Q_47 → Q_0
Q_{0-46} → Q_{1-47}
EXIT
CENTRAL PROCESSOR HALTS

Jump Instructions
Micro-Flow Chart



† When CI is controlled by an RPT or DR instruction, $[NI]_0 = [RPT]_0$ or $[DR]_0$

INSTRUCTION	CONDITION FOR TRANSFER OF CONTROL	INSTRUCTION	CONDITION FOR TRANSFER OF CONTROL
JMPL, JMPR	UNCONDITIONAL	JQNL, JQNR	$Q_0 = 1$
JAZL, JAZR	(A) = 0	JQEL, JQER	$Q_{47} = 0$
JNOL, JNOR	OVF = 0	JQOL, JQOR	$Q_{47} = 1$
JOFL, JOFR	OVF = 1	JDPL, JDPR	$D_0 = 0$
JAPL, JAPR	$A_0 = 0$	JAGQFL, JAGQFR	A ≥ Q FLOATING-POINT
JANL, JANR	$A_0 = 1$	JAGQL, JAGQR	A ≥ Q FIXED-POINT
JAEDL, JAEDR	(A) = (D)	JAGDL, JAGDR	A ≥ D ALPHANUMERIC
JAEQL, JAEQR	(A) = (Q)	JL, JR	UNCONDITIONAL
JQPL, JQPR	$Q_0 = 0$	JBTL, JBTR	SETTING OF BREAKPOINT SWITCHES

JNOL **0202** *Jump if No Overflow*

JNOR **2202**

The JNO instruction is executed as a JMP instruction *if the Overflow Indicator equals zero (no overflow indicated).* If the Indicator equals one (overflow), the instruction immediately following the JNO instruction is executed. This instruction clears the Overflow Indicator to zero even if an ICOS instruction has been given and is still in effect.

JOFL **0203** *Jump If Overflow*

JOFR **2203**

The JOF instruction is executed as a JMP instruction *if the Overflow Indicator equals one (overflow indicated).* If the indicator equals zero (no overflow), the instruction immediately following the JOF instruction is executed. This instruction clears the Overflow Indicator to zero even if an ICOS instruction has been given and is still in effect.

JAPL **0210** *Jump If the Contents of A are Positive*

JAPR **2210**

The JAP instruction is executed as a JMP instruction *if the leftmost bit of the A Register is zero.* If the bit is *not* zero, the instruction immediately following the JAP instruction is executed.

JANL **0211** *Jump If the Contents of A are Negative*

JANR **2211**

The JAN instruction is executed as a JMP instruction *if the leftmost bit of the A Register is one.* If the bit is *not* one, the instruction immediately following the JAN instruction is executed.

JAEDL **0213** *Jump If A Equals D*

JAEDR **2213**

The JAED instruction is executed as a JMP instruction *if the contents of the A Register are equal to the contents of the D Register.* Otherwise, the instruction immediately following the JAED instruction is executed.

JAEQL 0212 *Jump If A Equals Q*
JAEQR 2212

The JAEQ instruction transfers the contents of the Q Register to the D Register and then is executed as a JAED instruction. The word from Q remains in D.

JQPL 0220 *Jump If Q is Positive*
JQPR 2220

The JQP instruction is executed as a JMP instruction *if the leftmost bit of the Q Register is zero.* If the bit is *not* zero, the instruction immediately following the JQP instruction is executed. In either case, the contents of the Q Register are shifted circularly one bit to the left. (The bits in the Q Register are rotated one bit to the left. Bit zero enters bit 47.) Overflow is ignored.

JQNL 0221 *Jump If Q is Negative*
JQNR 2221

The JQN instruction is executed as a JMP instruction *if the leftmost bit of the Q Register is one.* If the leftmost bit is zero, the instruction immediately following the JQN instruction is executed. In either case, the contents of the Q Register are shifted circularly one bit to the left. Overflow is ignored.

JQEL 0222 *Jump If Q is Even*
JQER 2222

The JQE instruction is executed as a JMP instruction *if the rightmost bit of the Q Register is zero.* If the rightmost bit is *not* zero, the instruction immediately following the JQE instruction is executed. In either case, the contents of the Q Register are shifted circularly one bit to the right.

JQOL 0223 *Jump If Q is Odd*
JQOR 2223

The JQO instruction is executed as a JMP instruction *if the rightmost bit of the Q Register is one.* If the rightmost bit is *not* one, the instruction immediately following the JQO instruction is executed. In either case, the contents of the Q Register are shifted circularly one bit to the right.

JDPL 0230 *Jump If D is Positive*
JDPR 2230

The JDP instruction is executed as a JMP instruction *if the leftmost bit of the D Register is zero.* If the leftmost bit is *not* zero, the instruction immediately following the JDP instruction is executed.

JAGQFL 0231 *Jump If A is Greater than or equal to Q, Floating Point*
JAGQFR 2231

The JAGQF instruction transfers the word in the Q Register to the D Register. If the word in the A Register is *greater than or equal to* the word in the D Register, the JAGQF instruction is executed as a JMP instruction. If the word in the A Register is less than the word in the D Register, the next sequential instruction is executed. Both words are compared as floating-point numbers. The word originally in Q remains in D.

JAGQL 0232 *Jump If A is Greater than or equal to Q*
JAGQR 2232

The JAGQ instruction is the same as JAGQFL, except that the contents of the registers are considered to be signed, fixed point numbers.

JAGDL 0233 *Jump If A is Greater than or equal to D*
JAGDR 2233

The JAGD instruction is executed as a JMP instruction *if the contents of the A Register are greater than or equal to* the contents of the D Register. If the contents of the A Register are not greater than or equal to the contents of the D Register, the instruction immediately following the JAGD instruction is executed. The words in both registers are considered as alphanumeric words (see page 15) and are compared bit-by-bit.

JL 0320 *Jump*
JR 2320

The J instruction is executed as a JMP instruction except that the location of the next instruction is *not* stored in the JA Register.

JBTL 0001 *Breakpoint Jump*

JBTR 2001

The JBT instruction is executed in one of four ways depending on the setting of two adjacent, two-state pushbuttons on the Operator's Console, the Breakpoint Jump and Breakpoint Halt buttons. The setting of these buttons and the operations they effect are shown in the following table:

Breakpoint Jump	Breakpoint Halt	Operation
On	On	Jump after Halt
On	Off	Unconditional Jump
Off	On	Halt
Off	Off	Execute next instruction

The Breakpoint Jump and Halt switches cannot be operated and their status cannot be changed while the computer is cycling.

SHIFT INSTRUCTIONS

Shift Instructions shift all or some of the bits in a register right, left or circularly.

All Shift instructions first clear the Overflow Indicator to zero unless an ICOS instruction has been given and is still in effect.

SLAQ 2100 *Shift Left A and Q*

$$A_{1-47} \rightarrow A_{0-46}$$

$$Q_0 \rightarrow A_{47}$$

$$Q_{1-47} \rightarrow Q_{0-46}$$

$$0 \rightarrow Q_{47}$$

The SLAQ instruction shifts the bits of the A and Q Registers (considered as one 96-bit word) left the number of bits, modulo 64, specified in the V-field of the instruction. The bits shifted out of the left side of the A Register are lost and vacated bits at the right side of the Q Register are replaced by zeros. Fixed-point overflow will occur if the sign bit is changed or a significant bit is lost.

SRAQ **2101** *Shift Right A and Q*

$0 \rightarrow A_0$

$A_{0\text{-}46} \rightarrow A_{1\text{-}47}$

$A_{47} \rightarrow Q_0$

$Q_{0\text{-}46} \rightarrow Q_{1\text{-}47}$

The SRAQ instruction shifts the bits of the A and Q Registers (considered as one 96-bit word) right the number of bits, modulo 64, specified in the V-field of the instruction. The bits shifted out of the right side of the Q Register are lost, and the vacated bits at the left side of the A Register are replaced by zeros.

SLAQN **2102** *Shift Left A and Q Numerical*

$A_0 \rightarrow A_0$

$A_{2\text{-}47} \rightarrow A_{1\text{-}46}$

$Q_1 \rightarrow A_{47}$

$Q_0 \rightarrow Q_0$

$Q_{2\text{-}47} \rightarrow Q_{1\text{-}46}$

$0 \rightarrow Q_{47}$

The SLAQN instruction is the same as an SLAQ instruction, except that the sign bits of the A and Q Registers are not shifted or changed. The bits are shifted out of Q at bit one. Fixed-point overflow will occur if a significant bit is lost.

SRAQN **2103** *Shift Right A and Q Numerical*

$A_0 \rightarrow A_0$

$A_0 \rightarrow A_1$

$A_{1\text{-}46} \rightarrow A_{2\text{-}47}$

$Q_0 \rightarrow Q_0$

$A_{47} \rightarrow Q_1$

$Q_{1\text{-}46} \rightarrow Q_{2\text{-}47}$

The SRAQN instruction is the same as the SRAQ instruction, except that the sign bits of the A and Q Registers are not shifted or changed and the sign bit of A is propagated in the vacated bits to its right, i.e., if the sign bit is one, a one is inserted in all the vacated bits. The bits shifted out of A enter Q, in order, at bit one.

SLA 2110 *Shift Left A*

$$A_{1\text{-}47} \rightarrow A_{0\text{-}46}$$
$$0 \rightarrow A_{47}$$

The SLA instruction shifts the bits of the A Register left the number of bits, modulo 64, specified in the V-field of the instruction. The bits shifted out of the left side of the A Register are lost, and the vacated bits at the right side are replaced by zeros. Fixed-point overflow will occur if the sign bit is changed or a significant bit is lost.

SRA 2111 *Shift Right A*

$$0 \rightarrow A_0$$
$$A_{0\text{-}46} \rightarrow A_{1\text{-}47}$$

The SRA instruction shifts the bits of the A Register right the number of bits, modulo 64, specified in the V-field of the instruction. The bits shifted out of the right side of the A Register are lost, and the vacated bits at the left side are replaced by zeros.

SLAN 2112 *Shift Left A Numerically*

$$A_0 \rightarrow A_0$$
$$A_{2\text{-}47} \rightarrow A_{1\text{-}46}$$
$$0 \rightarrow A_{47}$$

The SLAN instruction is the same as the SLA instruction except that the sign bit of the A Register is not shifted or changed. Fixed-point overflow will occur if a significant bit is lost.

SRAN 2113 *Shift Right A Numerically*

$$A_0 \rightarrow A_0$$
$$A_0 \rightarrow A_1$$
$$A_{1\text{-}46} \rightarrow A_{2\text{-}47}$$

The SRAN instruction is the same as the SRA instruction, except that the sign bit of the A Register is not shifted or changed and is propagated in the vacated bits to its right.

SLQ 2120 *Shift Left Q*

$$Q_{1\text{-}47} \rightarrow Q_{0\text{-}46}$$
$$0 \rightarrow Q_{47}$$

The SLQ instruction shifts the bits of the Q Register left the number of bits, modulo 64, specified in the V-field of the instruction. The bits at the left side of the Q Register are lost, and the vacated bits at the right side are replaced by zeros. Fixed-point overflow will occur if the sign bit is changed or if a significant bit is lost.

SRQ 2121 *Shift Right Q*

$$0 \rightarrow Q_0$$
$$Q_{0\text{-}46} \rightarrow Q_{1\text{-}47}$$

The SRQ instruction shifts the bits of the Q Register right the number of bits, modulo 64, specified in the V-field of the instruction. The bits at the right side of the Q Register are lost, and the vacated bits at the left side are replaced by zeros.

SLQN 2122 *Shift Left Q Numerically*

$$Q_0 \rightarrow Q_0$$
$$Q_{2\text{-}47} \rightarrow Q_{1\text{-}46}$$
$$0 \rightarrow Q_{47}$$

The SLQN instruction is the same as the SLQ instruction except that the sign bit of the Q Register is not shifted or changed. Fixed-point overflow will occur if a significant bit is lost.

SRQN 2123 *Shift Right Q Numerically*

$$Q_0 \rightarrow Q_0$$
$$Q_0 \rightarrow Q_1$$
$$Q_{1\text{-}46} \rightarrow Q_{2\text{-}47}$$

The SRQN instruction is the same as the SRQ instruction, except that the sign bit of the Q Register is not shifted or changed, and is propagated in the vacated bits to its right.

SCD 2130 *Shift Circular D*

SCD 2132

$$D_{47} \rightarrow D_0$$

$$D_{0-46} \rightarrow D_{1-47}$$

The SCD instruction rotates the bits of the D Register right the number of bits, modulo 64, specified in the V-field of the instruction. Bits shifted out of the right side of the D Register are returned, in order, at the left side of D.

SRD 2131 *Shift Right D*

$$0 \rightarrow D_0$$

$$D_{0-46} \rightarrow D_{1-47}$$

The SRD instruction shifts the bits of the D Register right the number of bits, modulo 64, specified in the V-field of the instruction. The bits shifted out of the right side of the D Register are lost, and the vacated bits at the left side are replaced by zeros.

INDEX

SRDN 2133 *Shift Right D Numerically*

$$D_0 \rightarrow D_0$$

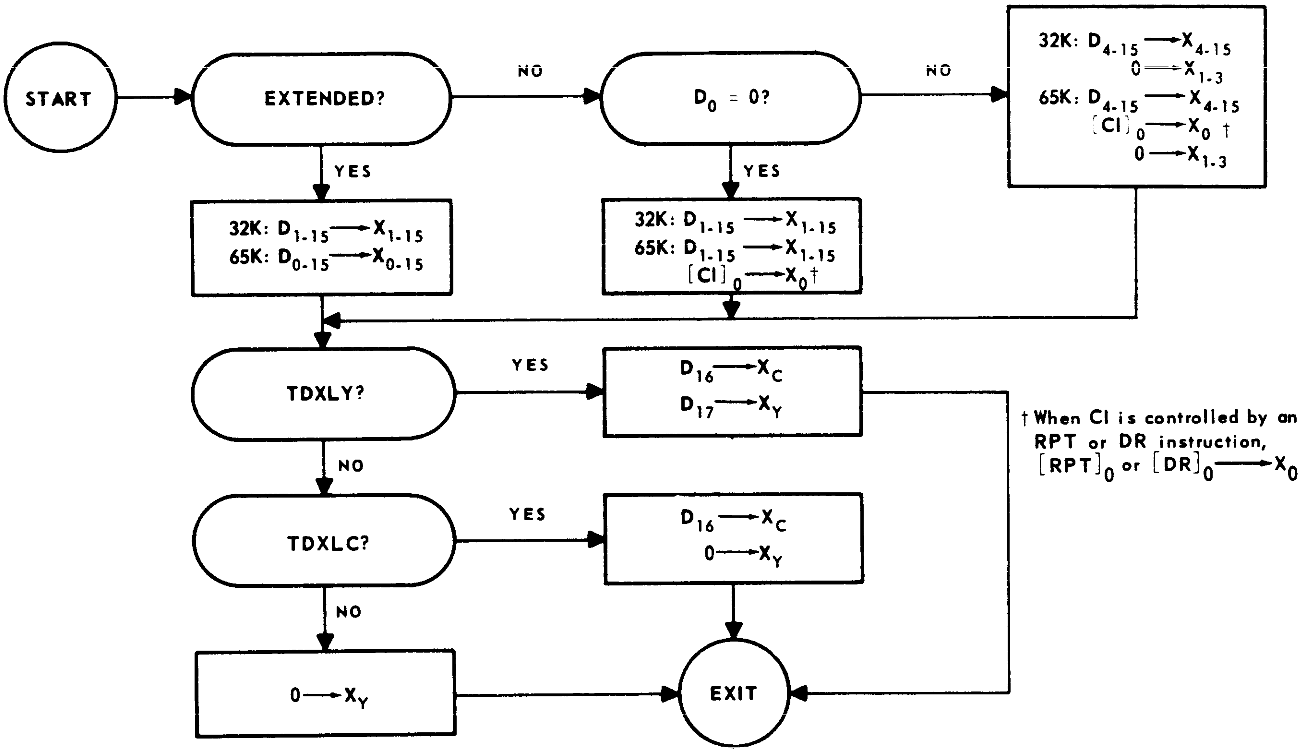
$$D_0 \rightarrow D_1$$

$$D_{1-46} \rightarrow D_{2-47}$$

The SRDN instruction is the same as SRD instruction, except that the sign bit of the D Register is not shifted or changed and is propagated in the bits to its right.

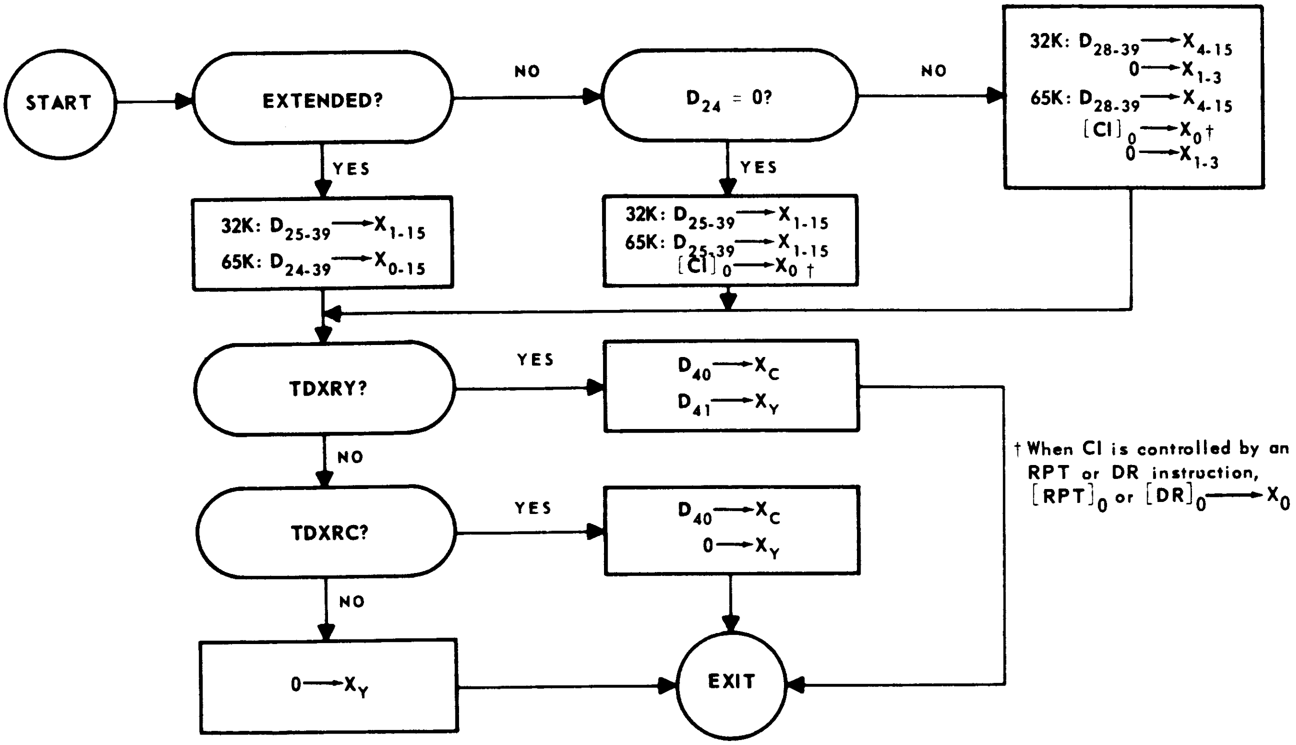
INDEX REGISTER INSTRUCTIONS Index Register Instructions alter the contents of an index register, set the C-bits and Y-bits of an index register to one or zero, or transfer information between an index register and the D Register.

An Index Register Instruction may operate on an address field in the D Register or on its own address field. The number of bits involved in such an operation depends on the size of memory and on the setting of the S-bit of the address field operated on, unless the instruction is extended by an EXT instruction (see pages 88-92).


START
EXTENDED?
NO
YES
$D_0 = 0?$
NO
YES
32K: $D_{4\text{-}15} \longrightarrow X_{4\text{-}15}$
$0 \longrightarrow X_{1\text{-}3}$
65K: $D_{4\text{-}15} \longrightarrow X_{4\text{-}15}$
$[CI]_0 \longrightarrow X_0$ †
$0 \longrightarrow X_{1\text{-}3}$
32K: $D_{1\text{-}15} \longrightarrow X_{1\text{-}15}$
65K: $D_{0\text{-}15} \longrightarrow X_{0\text{-}15}$
32K: $D_{1\text{-}15} \longrightarrow X_{1\text{-}15}$
65K: $D_{1\text{-}15} \longrightarrow X_{1\text{-}15}$
$[CI]_0 \longrightarrow X_0$†
TDXLY?
YES
$D_{16} \longrightarrow X_C$
$D_{17} \longrightarrow X_Y$
NO
TDXLC?
YES
$D_{16} \longrightarrow X_C$
$0 \longrightarrow X_Y$
NO
$0 \longrightarrow X_Y$
EXIT

† When CI is controlled by an RPT or DR instruction, $[RPT]_0$ or $[DR]_0 \longrightarrow X_0$

TDXL, TDXLC, TDXLY
Micro-Flow Chart


START
EXTENDED?
NO
YES
$D_{24} = 0?$
NO
YES
32K: $D_{28\text{-}39} \longrightarrow X_{4\text{-}15}$
$0 \longrightarrow X_{1\text{-}3}$
65K: $D_{28\text{-}39} \longrightarrow X_{4\text{-}15}$
$[CI]_0 \longrightarrow X_0$†
$0 \longrightarrow X_{1\text{-}3}$
32K: $D_{25\text{-}39} \longrightarrow X_{1\text{-}15}$
65K: $D_{24\text{-}39} \longrightarrow X_{0\text{-}15}$
32K: $D_{25\text{-}39} \longrightarrow X_{1\text{-}15}$
65K: $D_{25\text{-}39} \longrightarrow X_{1\text{-}15}$
$[CI]_0 \longrightarrow X_0$ †
TDXRY?
YES
$D_{40} \longrightarrow X_C$
$D_{41} \longrightarrow X_Y$
NO
TDXRC?
YES
$D_{40} \longrightarrow X_C$
$0 \longrightarrow X_Y$
NO
$0 \longrightarrow X_Y$
EXIT

† When CI is controlled by an RPT or DR instruction, $[RPT]_0$ or $[DR]_0 \longrightarrow X_0$

TDXR, TDXRC, TDXRY
Micro-Flow Chart

TDXL 0300 *Transfer D to Index Register*
TDXR 2300

The TDX instruction transfers the V-field of the specified half (L or R for left or right) of the word in the D Register to a specified index register. The Y-bit of the specified index register is set to zero.

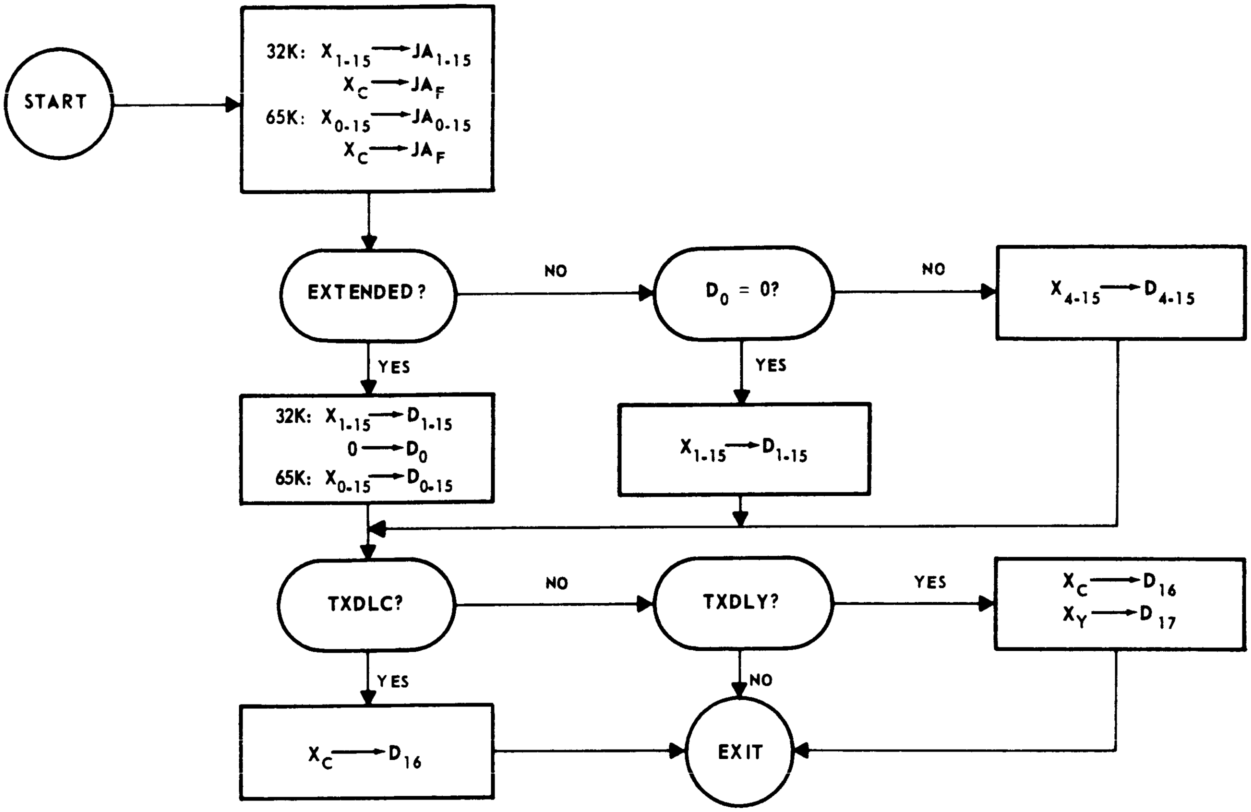
On a 65K system, if this instruction is not extended by an EXT instruction (see pages 88-92), the first bit of the specified index register is set equal to the first bit of the 16-bit address of the memory location in which the instruction is stored, unless this instruction is controlled by an RPT or DR instruction and is not extended. If this instruction is controlled by an RPT or DR instruction and is not extended, the first bit of the index register is set equal to the first bit of the 16-bit address of the memory location in which the RPT or DR instruction is stored.

TDXLC 0301 *Transfer D to Index Register with C-Bit*
TDXRC 2301

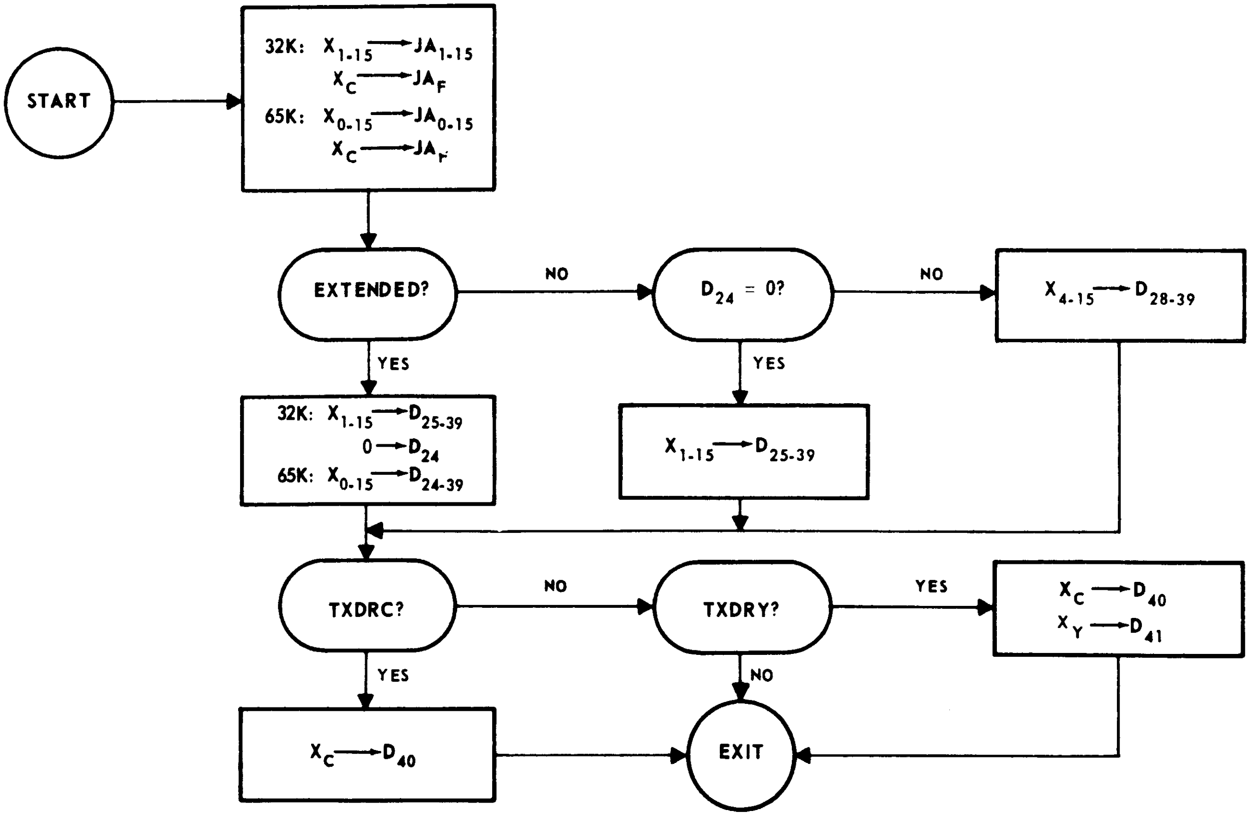
The TDXLC/TDXRC instruction is the same as the TDX instruction, except that the C-bit of the index register is replaced by the F-bit of the specified half of the word in the D Register (see page 16). The Y-bit of the specified index register is set to zero.

TDXLY 0322 *Transfer D to Index Register with C-Bit and Y-Bit*
TDXRY 2322

The TDXLY/TDXRY instruction is the same as the TDX instruction, except that the C-bit and the Y-bit of the index register are replaced by bits 16 and 17 of D (if TDXLY), or by bits 40 and 41 of D (if TDXRY).


START
32K: $X_{1-15} \rightarrow JA_{1-15}$
$X_C \rightarrow JA_F$
65K: $X_{0-15} \rightarrow JA_{0-15}$
$X_C \rightarrow JA_F$
EXTENDED?
NO
$D_0 = 0$?
NO
$X_{4-15} \rightarrow D_{4-15}$
YES
32K: $X_{1-15} \rightarrow D_{1-15}$
$0 \rightarrow D_0$
65K: $X_{0-15} \rightarrow D_{0-15}$
YES
$X_{1-15} \rightarrow D_{1-15}$
TXDLC?
NO
TXDLY?
YES
$X_C \rightarrow D_{16}$
$X_Y \rightarrow D_{17}$
YES
NO
$X_C \rightarrow D_{16}$
EXIT

TXDL, TXDLC, TXDLY
Micro-Flow Chart


START
32K: $X_{1-15} \rightarrow JA_{1-15}$
$X_C \rightarrow JA_F$
65K: $X_{0-15} \rightarrow JA_{0-15}$
$X_C \rightarrow JA_F$
EXTENDED?
NO
$D_{24} = 0$?
NO
$X_{4-15} \rightarrow D_{28-39}$
YES
32K: $X_{1-15} \rightarrow D_{25-39}$
$0 \rightarrow D_{24}$
65K: $X_{0-15} \rightarrow D_{24-39}$
YES
$X_{1-15} \rightarrow D_{25-39}$
TXDRC?
NO
TXDRY?
YES
$X_C \rightarrow D_{40}$
$X_Y \rightarrow D_{41}$
YES
NO
$X_C \rightarrow D_{40}$
EXIT

TXDR, TXDRC, TXDRY
Micro-Flow Chart

TXDL 0302 *Transfer from Index Register to D*
TXDR 2302

The TXD instruction transfers the contents of a specified index register to the JA Register, then transfers 12 or 15 bits from JA to the specified half (L or R for left or right) of the D Register depending on the setting of the S-bit in D, unless this instruction is extended by an EXT instruction (see pages 88-92). Only the specified address field of D is affected. The entire field from the index register remains in JA. The C-bit is transferred to the F-bit of JA, remaining there; it is not transferred to the D Register.

INDEX

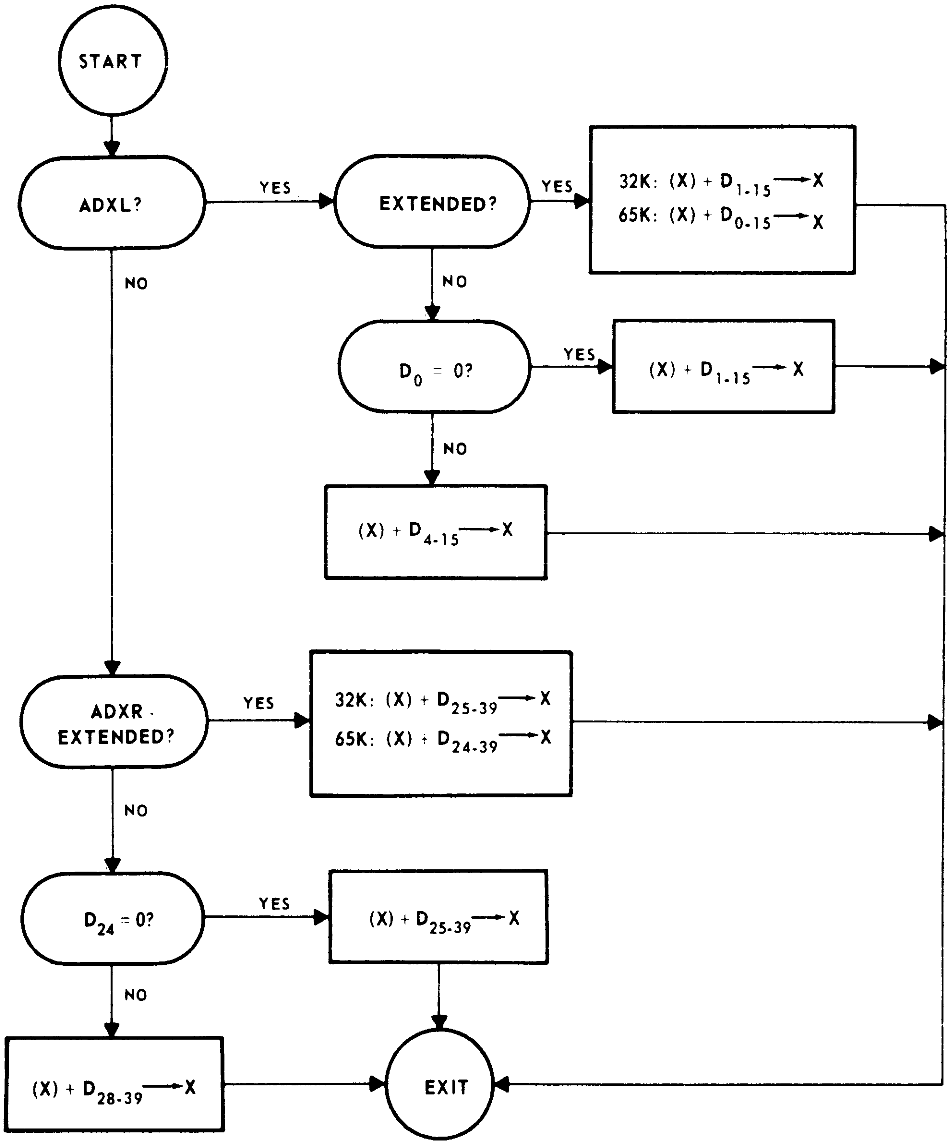
TXDLC 0303 *Transfer from Index Register to D with C-Bit*
TXDRC 2303

The TXDLC/TXDRC instruction is the same as the TXD instruction, except that the C-bit which was transferred to the F-bit of the JA Register is also transferred to the F-bit corresponding to the specified half (L or R for left or right) of the D Register.

TXDLY 0323 *Transfer from Index Register to D with C-Bit and Y-Bit*
TXDRY 2323

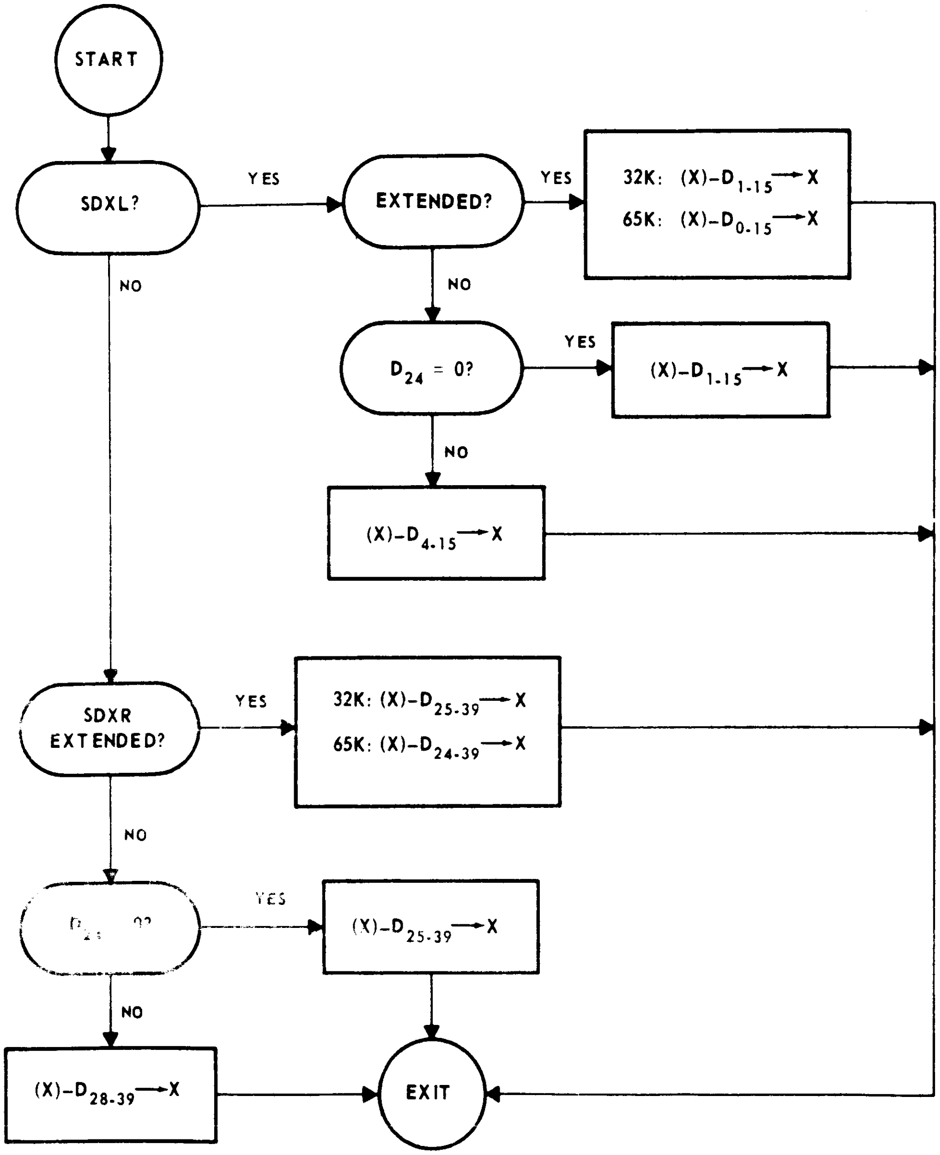
The TXDLY/TXDRY instruction is the same as the TXDL/TXDR instruction, except that C-bit and the Y-bit are transferred to bits 16 and 17 of D (TXDLY) or to bits 40 and 41 of D (TXDRY).

ADXL 0310 *Add D to Index Register*
ADXR 2310



The ADX instruction adds the V-field of the specified half (L or R for left or right) of the D Register to the contents of a specified index register. (If this instruction is extended by an EXT instruction, see pages 88-92.) The sum which replaces the original contents of the index register is modulo memory size. No overflow is indicated.

SDXL 0311 *Subtract D from Index Register*
SDXR 2311

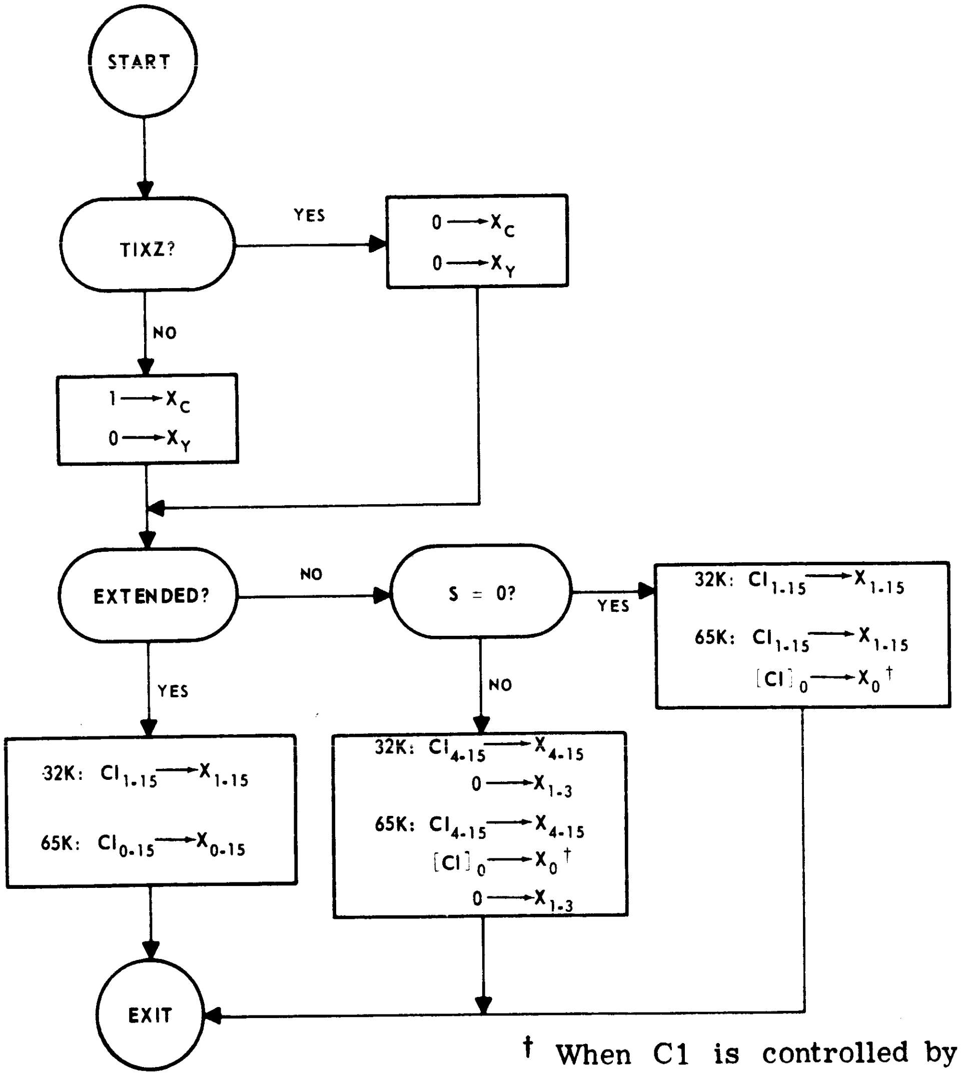

START
SDXL?
YES
EXTENDED?
YES
32K: (X)-$D_{1\text{-}15}$ → X
65K: (X)-$D_{0\text{-}15}$ → X
NO
NO
D_{24} = 0?
YES
(X)-$D_{1\text{-}15}$ → X
NO
(X)-$D_{4\text{-}15}$ → X
SDXR
EXTENDED?
YES
32K: (X)-$D_{25\text{-}39}$ → X
65K: (X)-$D_{24\text{-}39}$ → X
NO
YES
(X)-$D_{25\text{-}39}$ → X
NO
(X)-$D_{28\text{-}39}$ → X
EXIT

The SDX instruction subtracts the V-field of the left (if SDXL) or right (SDXR) half of the D Register from the contents of a specified index register. (If this instruction is extended by an EXT instruction, see pages 88-92.) The difference which replaces the original contents of the index register is modulo memory size. No overflow is indicated.

REGISTER

TIXZ **0321** *Transfer Instruction Address Field to Index Register*

TIXS **2321**


START
TIXZ?
YES
$0 \longrightarrow X_C$
$0 \longrightarrow X_Y$
NO
$1 \longrightarrow X_C$
$0 \longrightarrow X_Y$
EXTENDED?
NO
S = 0?
YES
32K: $CI_{1\text{-}15} \longrightarrow X_{1\text{-}15}$
65K: $CI_{1\text{-}15} \longrightarrow X_{1\text{-}15}$
$[CI]_0 \longrightarrow X_0$ †
YES
32K: $CI_{1\text{-}15} \longrightarrow X_{1\text{-}15}$
65K: $CI_{0\text{-}15} \longrightarrow X_{0\text{-}15}$
NO
32K: $CI_{4\text{-}15} \longrightarrow X_{4\text{-}15}$
$0 \longrightarrow X_{1\text{-}3}$
65K: $CI_{4\text{-}15} \longrightarrow X_{4\text{-}15}$
$[CI]_0 \longrightarrow X_0$ †
$0 \longrightarrow X_{1\text{-}3}$
EXIT

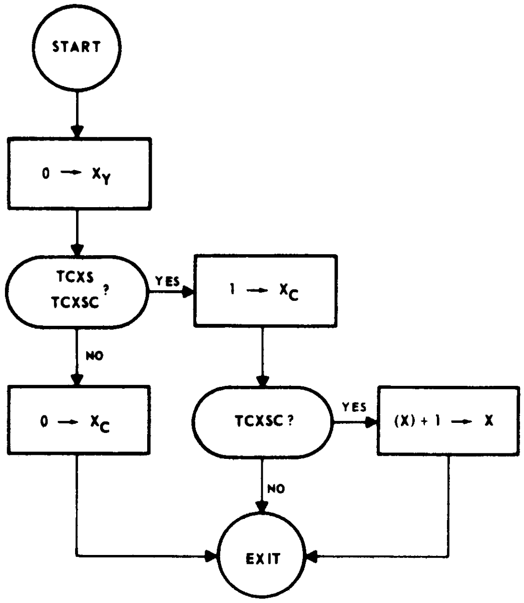
† When C1 is controlled by an RPT or DR instruction $[\text{RPT}]_0$ or $[\text{DR}]_0 \longrightarrow X_0$

The TIX instruction transfers its V-field to a specified index register. For TIXS, the C-bit is set to one; for TIXZ, the C-bit is set to zero. This instruction sets the Y-bit of the specified index register to zero.

On a 65K system, if this instruction is not extended by an EXT instruction (see pages 88-92), the first bit of the index register is set to the value of the first bit of the 16-bit address of the memory location in which the instruction is stored, unless this instruction is controlled by an RPT or DR instruction. If this instruction is controlled by an RPT or DR instruction and is not indexed, the first bit of the index register is set to the value of the first bit of the 16-bit address of the memory location in which the RPT or DR instruction is stored.

TCXZ	0013
TCXS	2013
TCXSC	2013

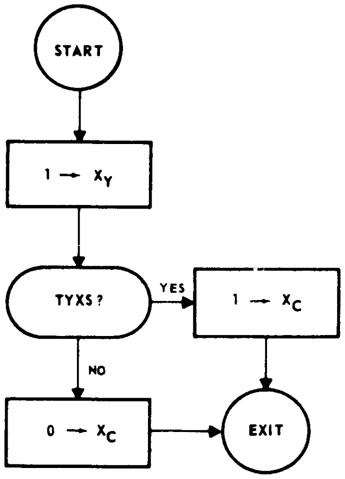
Transfer C-Bit to Index Register



The TCX instruction sets the C-bit of a specified index register to one if TCXS is written or to zero if TCXZ is written. TCXSC sets the C-bit to one and immediately increases the contents of the index register by one. The difference between the TCXS and TCXSC instructions is that the S-bit of the former is zero and the S-bit of the latter is one. This instruction sets the Y-bit of the specified index register to zero.

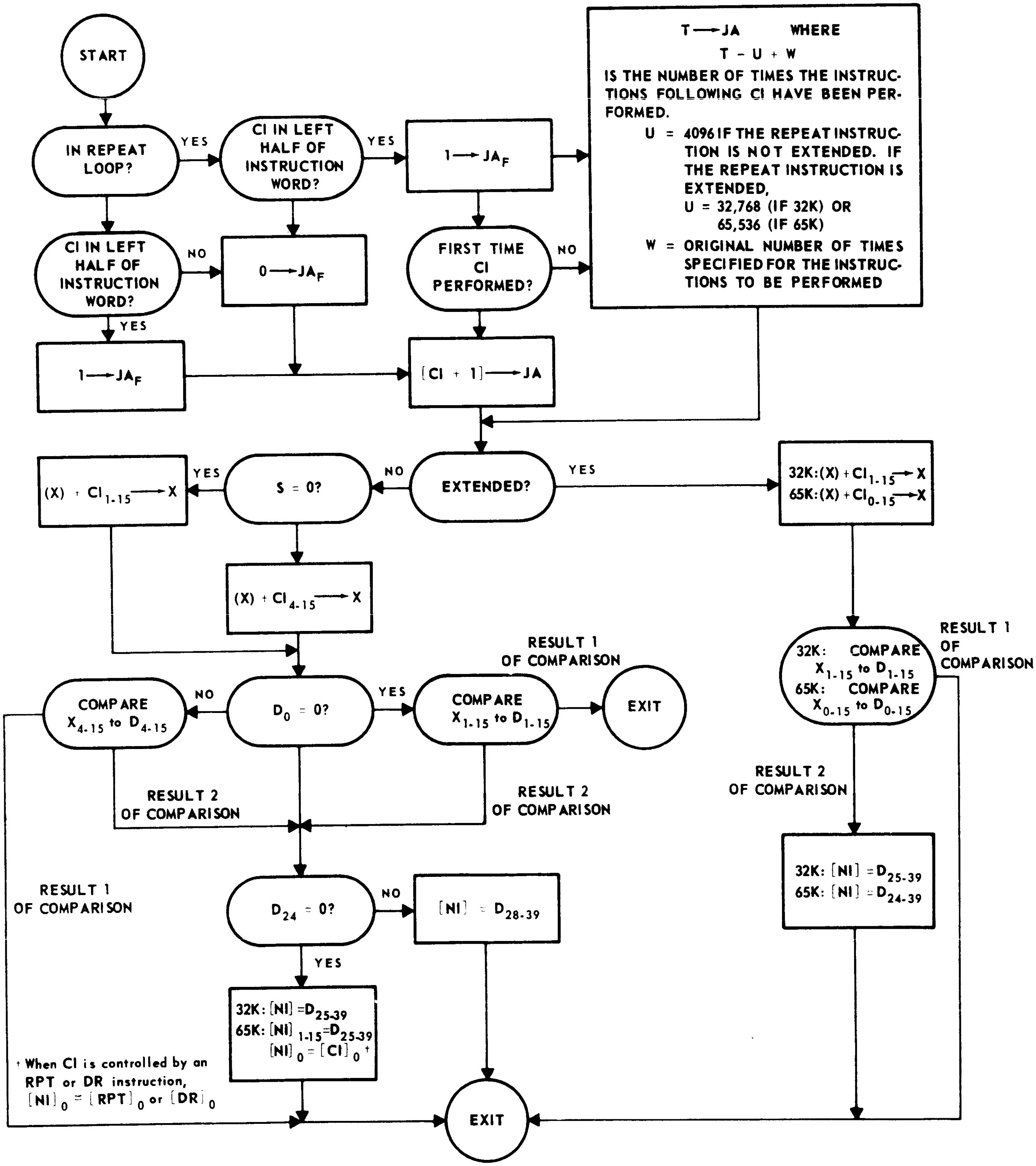
TYXZ	1332
TYXS	1333

Transfer Y-Bit to Index Register



The TYX instruction sets the C-bit of a specified index register to one if TYXS is written, or to zero if TYXZ is written. In both cases the Y-bit is set to one.

REGISTER


START
IN REPEAT LOOP?
YES
CI IN LEFT HALF OF INSTRUCTION WORD?
YES
1 → JA_F
$T \rightarrow JA$ WHERE
T - U + W
IS THE NUMBER OF TIMES THE INSTRUCTIONS FOLLOWING CI HAVE BEEN PERFORMED.
U = 4096 IF THE REPEAT INSTRUCTION IS NOT EXTENDED. IF THE REPEAT INSTRUCTION IS EXTENDED,
U = 32,768 (IF 32K) OR 65,536 (IF 65K)
W = ORIGINAL NUMBER OF TIMES SPECIFIED FOR THE INSTRUCTIONS TO BE PERFORMED
CI IN LEFT HALF OF INSTRUCTION WORD?
NO
0 → JA_F
FIRST TIME CI PERFORMED?
NO
YES
1 → JA_F
[CI + 1] → JA
EXTENDED?
NO
YES
S = 0?
YES
$(X) + CI_{1-15} \rightarrow X$
$(X) + CI_{4-15} \rightarrow X$
$32K{:}(X) + CI_{1-15} \rightarrow X$
$65K{:}(X) + CI_{0-15} \rightarrow X$
RESULT 1 OF COMPARISON
COMPARE X_{4-15} to D_{4-15}
NO
$D_0 = 0?$
YES
COMPARE X_{1-15} to D_{1-15}
EXIT
RESULT 1 OF COMPARISON
32K: COMPARE X_{1-15} to D_{1-15}
65K: COMPARE X_{0-15} to D_{0-15}
RESULT 2 OF COMPARISON
RESULT 2 OF COMPARISON
RESULT 2 OF COMPARISON
RESULT 1 OF COMPARISON
$D_{24} = 0?$
NO
$[NI] = D_{28-39}$
$32K{:}\ [NI] = D_{25-39}$
$65K{:}\ [NI] = D_{24-39}$
YES
$32K{:}\ [NI] = D_{25-39}$
$65K{:}\ [NI]_{1-15} = D_{25-39}$
$[NI]_0 = [CI]_0$ †
EXIT

† When CI is controlled by an RPT or DR instruction, $[NI]_0 = [RPT]_0$ or $[DR]_0$

INSTRUCTION	RESULT 1 OF COMPARISON	RESULT 2, OF COMPARISON
AIXJ	$X_{k-n} = D_{k-n}$	$X_{k-n} \neq D_{k-n}$
AIXJS	$X_{k-n} \geq D_{k-n}$	$X_{k-n} < D_{k-n}$
AIXJEG	$X_{k-n} < D_{k-n}$	$X_{k-n} \geq D_{k-n}$

AIXJ, AIXJS, AIXJEG Micro-Flow Chart

AIXJ 0330 *Add Instruction Address Field to Index Register and Jump*
AIXJ 2330

The AIXJ instruction places the address of the next sequential instruction in the JA Register (unless this instruction is in the left half of an instruction word controlled by an RPT or DR instruction), then adds the V-field of this AIXJ instruction to the contents of a specified index register. The sum is modulo memory size. No overflow is indicated. The original contents of the index register are replaced with the sum. The sum is then compared to the V-field of the left half of the word in the D Register. (If this instruction is extended by an EXT instruction, see page 88.) Only those bits in the index register that correspond with the V-field in D are used in the comparison. If the corresponding bits are *not equal,* control is transferred to the left or right half of the location specified by the V-field of the right half of the word in the D Register, depending on whether bit 40 of D is set to zero or one. If the corresponding bits are *equal,* the next sequential instruction is executed.

On a 65K system, if this instruction is not extended, equality of the first bit in the left half of D and the first bit of the specified index register is assumed. The value of the first bit of the 16-bit address of the memory location in which the instruction is stored is used as the first bit in a 16-bit jump address, unless this instruction is controlled by an RPT or DR instruction. If the instruction is controlled by an RPT or DR instruction and is not extended, the value of the first bit of the 16-bit address of the memory location in which the RPT or DR instruction is stored is used as the first bit in a 16-bit jump address.

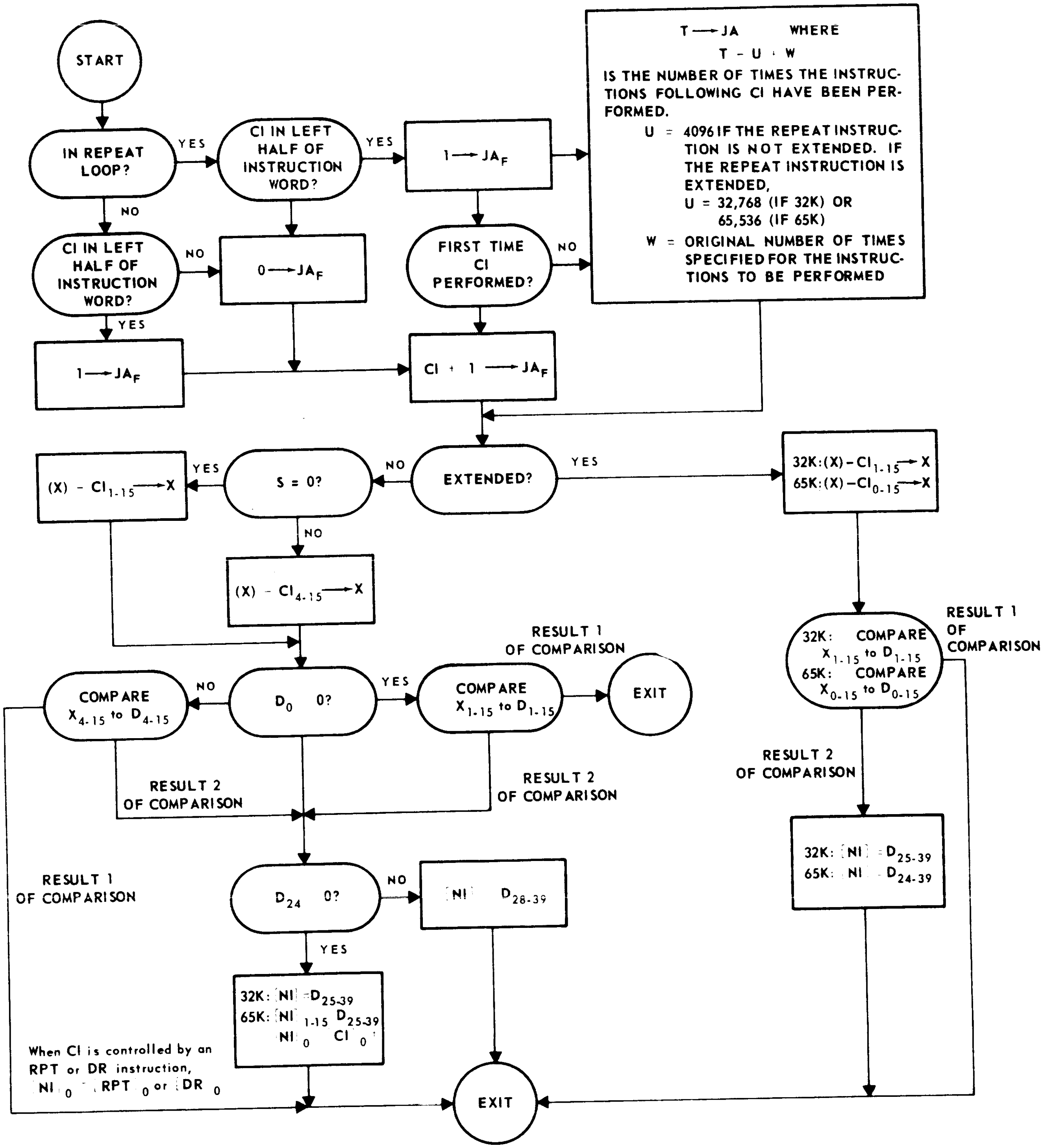
AIXJS 2312 *Add Instruction Address Field to Index Register and Jump if Smaller than D*

The AIXJS instruction is the same as an AIXJ instruction, except that the transfer of control is made only if the contents of the index register are *smaller than* the V-field of the left half of the word in D. Otherwise, the next sequential instruction is executed.

AIXJEG 0312 *Add Instruction Address Field to Index Register and Jump if Equal to or Greater than D*

The AIXJEG instruction is the same as the AIXJ instruction, except that the transfer of control is made only if the contents of the specified index register are *equal to* or *greater than* the V-field of the left half of D. Otherwise, the next sequential instruction is executed.

INDEX


START
IN REPEAT LOOP?
YES
CI IN LEFT HALF OF INSTRUCTION WORD?
YES
1 ⟶ JA_F
T ⟶ JA WHERE
T - U + W
IS THE NUMBER OF TIMES THE INSTRUCTIONS FOLLOWING CI HAVE BEEN PERFORMED.
U = 4096 IF THE REPEAT INSTRUCTION IS NOT EXTENDED. IF THE REPEAT INSTRUCTION IS EXTENDED,
U = 32,768 (IF 32K) OR 65,536 (IF 65K)
W = ORIGINAL NUMBER OF TIMES SPECIFIED FOR THE INSTRUCTIONS TO BE PERFORMED
NO
CI IN LEFT HALF OF INSTRUCTION WORD?
NO
0 ⟶ JA_F
FIRST TIME CI PERFORMED?
NO
YES
1 ⟶ JA_F
CI + 1 ⟶ JA_F
(X) - CI_{1-15} ⟶ X
YES
S = 0?
NO
EXTENDED?
YES
32K: (X) - CI_{1-15} ⟶ X
65K: (X) - CI_{0-15} ⟶ X
NO
(X) - CI_{4-15} ⟶ X
RESULT 1 OF COMPARISON
COMPARE X_{4-15} to D_{4-15}
NO
D_0 0?
YES
COMPARE X_{1-15} to D_{1-15}
EXIT
RESULT 1 OF COMPARISON
32K: COMPARE X_{1-15} to D_{1-15}
65K: COMPARE X_{0-15} to D_{0-15}
RESULT 2 OF COMPARISON
RESULT 2 OF COMPARISON
RESULT 2 OF COMPARISON
32K: [NI] = D_{25-39}
65K: NI - D_{24-39}
RESULT 1 OF COMPARISON
D_{24} 0?
NO
[NI] D_{28-39}
YES
32K: [NI] = D_{25-39}
65K: [NI] $_{1-15}$ D_{25-39}
NI $_0$ CI $_0$
When CI is controlled by an RPT or DR instruction,
NI $_0$ - RPT $_0$ or DR $_0$
EXIT

INSTRUCTION	RESULT 1 OF COMPARISON	RESULT 2 OF COMPARISON
SIXJ	$X_{k-n} = D_{k-n}$	$X_{k-n} \neq D_{k-n}$
SIXJG	$X_{k-n} \leq D_{k-n}$	$X_{k-n} > D_{k-n}$
SIXJES	$X_{k-n} < D_{k-n}$	$X_{k-n} \leq D_{k-n}$

SIXJ, SIXJG, SIXJES Micro-Flow Chart

SIXJ 0331 *Subtract Instruction Address Field from Index Register and Jump*

SIXJ 2331

The SIXJ instruction places the address of the next sequential instruction in the JA register (unless this instruction is in the left half of an instruction word controlled by an RPT or DR instruction), then subtracts the V-field of this SIXJ instruction from the contents of a specified index register. The difference is modulo memory size. The original contents of the index register are replaced by the difference, and the difference is compared with the V-field of the left half of the word in the D Register. (If this instruction is extended by an EXT instruction, see pages 88-92.) Only those bits in the index register that correspond with the V-field in D are used in the comparison. If the corresponding bits are *not equal,* control is transferred to the left or right half of the location specified by the V-field of the right half of the word in the D Register, depending on whether bit 40 of D is set to zero or one. If the corresponding bits are *equal,* the next sequential instruction is executed.

On a 65K system, if this instruction is not extended, equality of the first bit in the left half of D and the first bit of the specified index register is assumed. The value of the first bit of the 16-bit address of the memory location in which the instruction is stored is used as the first bit in a 16-bit jump address, unless this instruction is controlled by an RPT or DR instruction. If this instruction is controlled by an RPT or DR instruction and is not extended, the value of the first bit of the 16-bit address of the memory location in which the RPT or DR instruction is stored is used as the first bit in a 16-bit jump address.

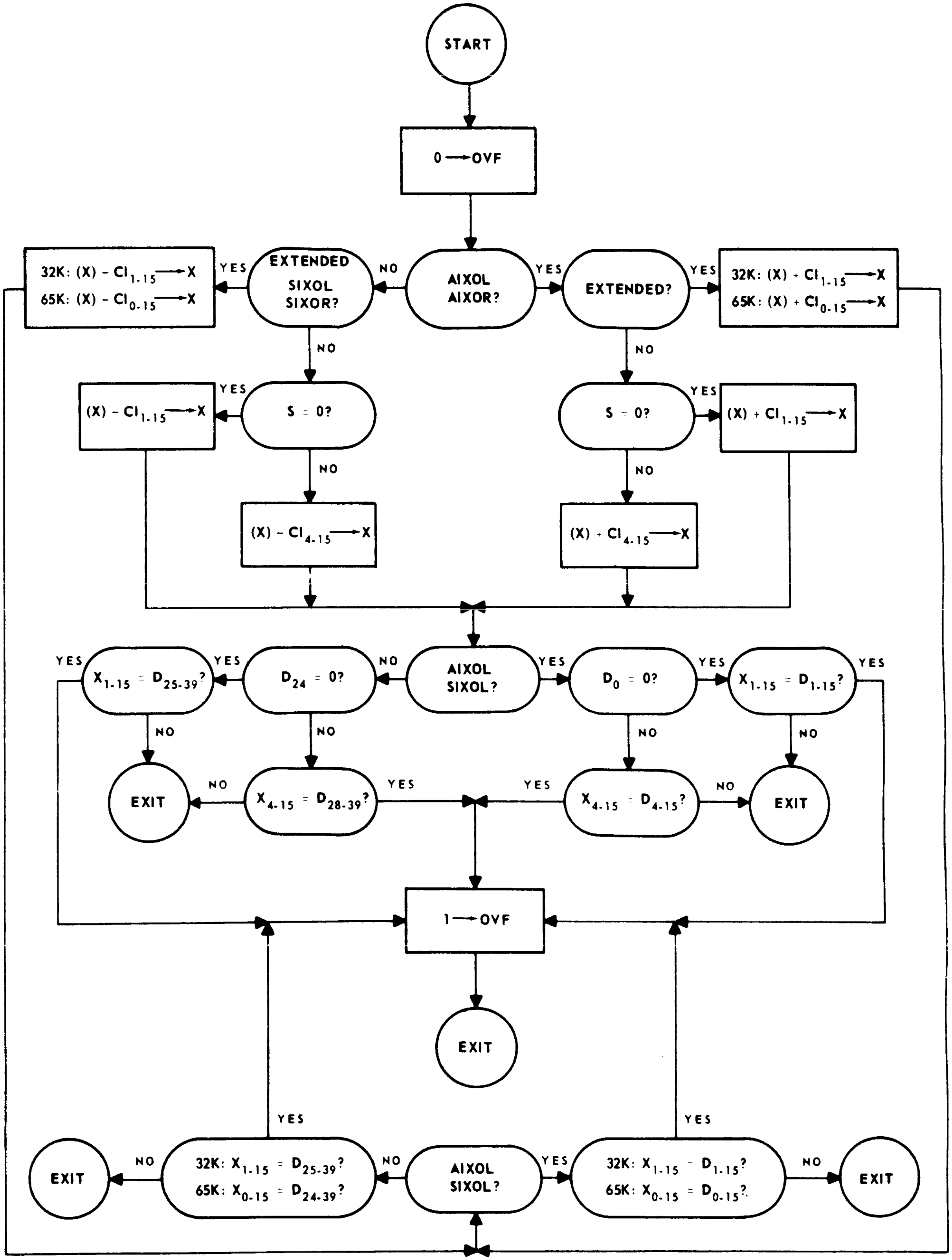
INDEX

SIXJG 0313 *Subtract Instruction Address Field from Index Register and Jump if Greater than D*

The SIXJG instruction is the same as the SIXJ instruction, except that the transfer of control is made only if the contents of the index register are *greater than* the V-field of the left half of the word in D. Otherwise, the next sequential instruction is executed.

SIXJES 2313 *Subtract Address Field from Index Register and Jump if Equal to or Smaller than D*

The SIXJES instruction is the same as the SIXJ instruction, except that the transfer of control is made only if the contents of the specified index register are *equal to* or *smaller than* the V-field of the left half of D. Otherwise, the next sequential instruction is executed.



AIXOL, AIXOR, SIXOL, SIXOR
Micro-Flow Chart

AIXOL **0332** *Add Instruction Address Field to Index Register and Set Overflow*
AIXOR **2332**

The AIXO instruction clears the Overflow Indicator to zero unless a previous ICOS instruction has been given and is still in effect. The V-field of this instruction is then added to the contents of a specified index register. The sum is modulo memory size. (No overflow is indicated.) The original contents of the index register are replaced with the sum, then the sum is compared to the V-field of the specified half (L or R for left or right) of the word in the D Register. (If this instruction is extended by an EXT instruction, see pages 88-92.) Only those bits in the index register that correspond with the V-field in D are used in the comparison. If the corresponding bits are *equal*, the Overflow Indicator is set to one.

On a 65K system, if this instruction is not extended by an EXT instruction, equality of the first bit in the specified half of D and the first bit in the specified index register is assumed.

INDEX

SIXOL **0333** *Subtract Instruction Address Field from Index Register*
SIXOR **2333** *and Set Overflow*

The SIXO instruction clears the Overflow Indicator to zero, unless a previous ICOS instruction has been given and is still in effect. The V-field of this instruction is then subtracted from the contents of a specified index register. The difference is modulo memory size. (No overflow is indicated.) The original contents of the index register are replaced with the difference, and the difference is then compared to the V-field of the specified half (L or R for left or right) of the word in the D Register. (If this instruction is extended by an EXT instruction, see pages 88-92.) Only those bits in the index register that correspond with the V-field in D are used in the comparison. If the corresponding bits are equal, the Overflow Indicator is set to one.

On a 65K system, if this instruction is not extended by an EXT instruction, equality of the first bit in the specified half of D and the first bit in the specified index register is assumed.

EXTRACT INSTRUCTIONS The Extract Instructions are bit-by-bit multiplications (i.e., there is no carry into adjacent bits) between the contents of Q and the corresponding bits of the designated memory location. This is a logical AND operation. The results of this operation, for the four possible bit configurations are:

```
0 1 0 1   Q REGISTER BEFORE AND AFTER EXTRACTION
0 0 1 1   MEMORY LOCATION BEFORE AND AFTER EXTRACTION (EXCEPT EIS)
-------
0 0 0 1   D REGISTER AFTER EXTRACTION
```

In the description below, the bits in the memory location specified, matched by one bits in the Q Register, are considered to be extracted fields.

ETD **0030** *Extract Transfer to D*

$$(Q) \cdot (M) \longrightarrow D$$

The ETD instruction extracts from the contents of the specified memory location according to the contents of the Q Register and transfers the extracted fields to the D Register. The remaining bits of D are set to zero.

ETA **2030** *Extract Transfer to A*

$$(Q) \cdot (M) \longrightarrow D$$
$$(Q) \cdot (M) \longrightarrow A$$

The ETA instruction extracts from the contents of the specified memory location according to the contents of the Q Register and transfers the extracted fields to the D Register and to the A Register. The A and D Registers receive and retain the extracted fields. The remaining bits of A and D are set to zero.

EI **0032** *Extract and Insert*

$$(M) \cdot (Q) \longrightarrow D$$
$$(A) \cdot (Q)' \vee (M) \cdot (Q) \longrightarrow A$$

The EI instruction extracts from the contents of the specified memory location according to the contents of the Q Register and inserts the extracted fields in the A Register without disturbing the remaining positions of the A Register. The D Register receives and retains the extracted fields. The remaining bits of D are set to zero.

EIS **2032** *Extract and Insert and Store*

$$(A) \cdot (Q)' \vee (M) \cdot (Q) \longrightarrow A, D \text{ and } M$$

The EIS instruction is the same as the EI instruction, except that the result in the A Register is transferred to D and to the specified memory location. The result replaces the contents of the specified memory location and of the A and D Registers.

EA **1322** *Extract and Add*
FEA **3322**

$$(Q) \cdot (M) \longrightarrow D$$
$$(A) + \{(Q) \cdot (M)\} \longrightarrow A$$

The EA/FEA instruction performs as an ETD instruction, then adds the extracted fields in D to the contents of the A Register. The sum replaces the contents of A, and the D Register retains the extracted field(s) except in the case of an FEA where the word in D (containing the extracted fields) had to be arranged for addition (see page 30). (The command FEA is for a floating-point addition.)

ES **1323** *Extract and Subtract*
FES **3323**

$$(Q) \cdot (M) \longrightarrow D$$
$$(A) - \{(Q) \cdot (M)\} \longrightarrow A$$

The ES/FES instruction performs as an ETD instruction, then subtracts the extracted fields from the contents of the A Register. The difference replaces the contents of A, and the D Register retains the extracted field(s) except in the case of an FES where the word in D (containing the extracted fields) had to be arranged for subtraction (see page 30). (The command FES is for a floating-point subtraction.)

LOGIC INSTRUCTIONS

Logic Instructions are bit-by-bit operations, including a logical inclusive OR and a logical exclusive OR.

DORMS 0031 *D Or Memory and Store*

$$(M) \vee (D) \longrightarrow D \text{ and } M$$

The DORMS instruction forms a composite word in D in which there are binary ones in every bit position for which there is a one in the D Register and/or in the specified memory location. The resulting word in D is then stored in the specified memory location. The A Register is not changed.

This is a logical inclusive OR operation. The results of this operation for the four possible bit configurations are:

0101	D REGISTER BEFORE EXECUTION
0011	MEMORY LOCATION BEFORE EXECUTION
0111	D REGISTER AND MEMORY LOCATION AFTER EXECUTION

AWCS 2031 *Add Without Carry and Store*

$$(A) \wedge (M) \longrightarrow D \text{ and } M$$

The AWCS instruction adds without carries the contents of the A Register to the contents of a specified memory location. The sum is placed in the D Register and transferred to the specified memory location. The A Register is not changed.

This is a logical exclusive OR operation. The results of this operation for the four possible bit configurations are:

0101	A REGISTER BEFORE AND AFTER EXECUTION
0011	MEMORY LOCATION EXECUTION
0110	MEMORY LOCATION AND D REGISTER AFTER EXECUTION

SPECIAL INSTRUCTIONS

HLTL 0000 *Halt*
HLTR 2000

The HLT instruction stops the Central Processor after modifying an index register, if index register modification is specified. When the Advance Bar on the Operator's Console is pressed, the Central Processor will proceed to the next sequential instruction.

NOPL 0003 *No Operation*
NOPR 2003

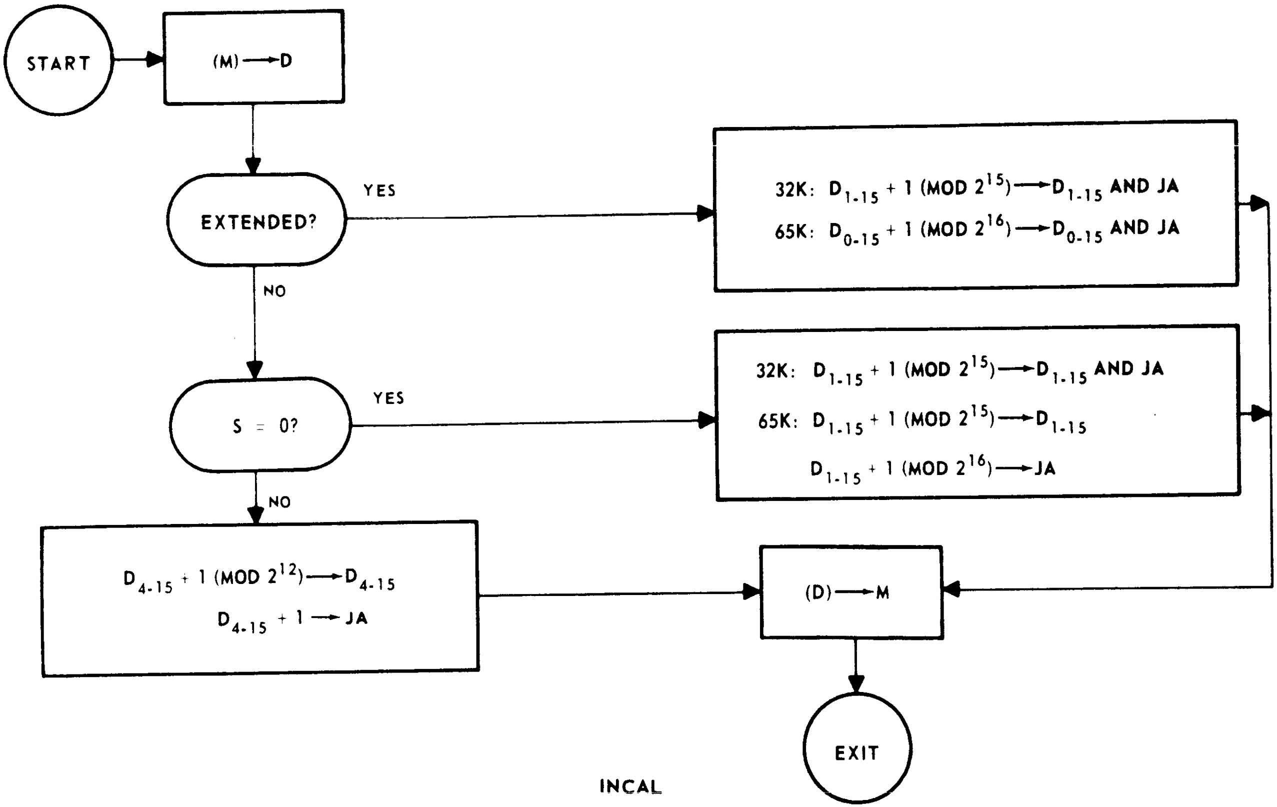
The NOP instruction causes the Central Processor to proceed to the next sequential instruction after modifying an index register, if index register modification is specified.

ICOS 2002 *Inhibit Clearing the Overflow Indicator*

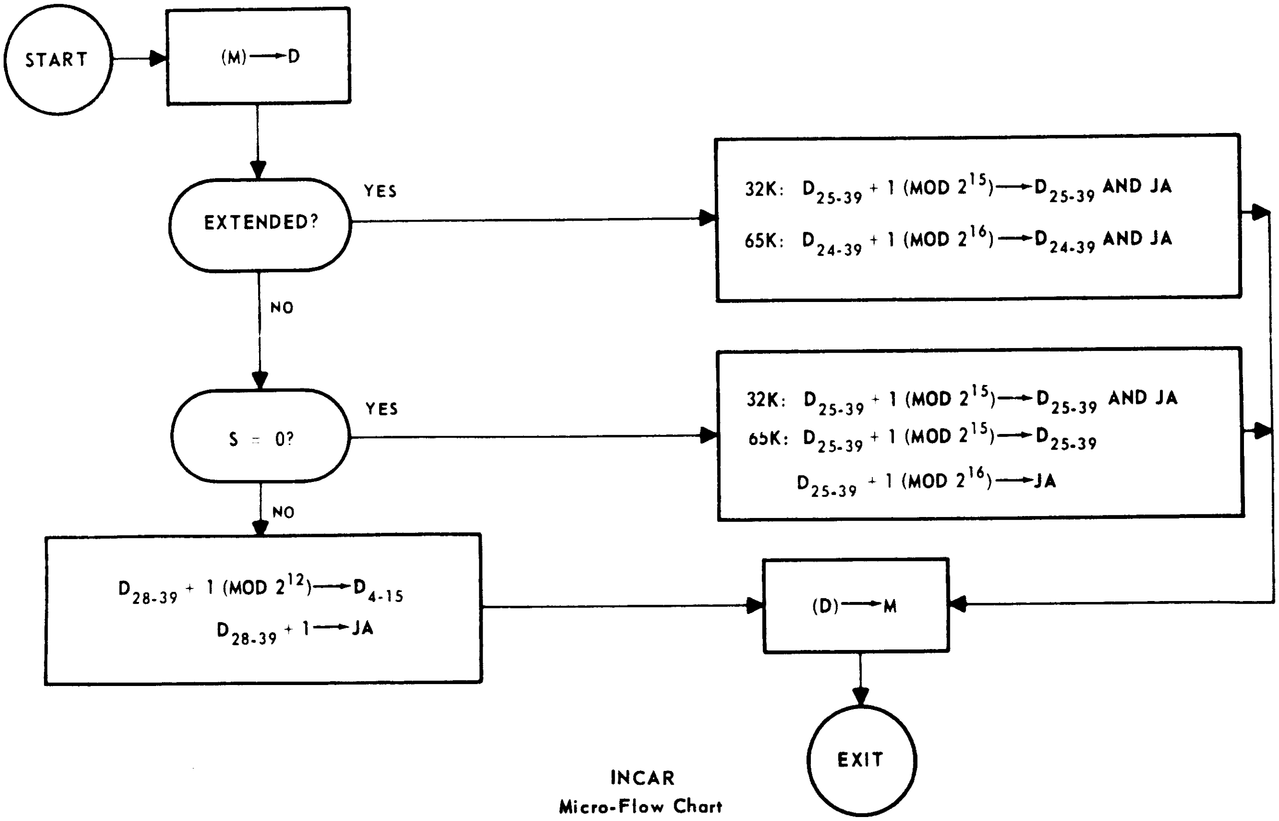
The ICOS instruction clears the Overflow Indicator to zero and inhibits its future clearing by arithmetic, Shift or Index Register instructions. (The JNO and JOF instructions clear the Overflow Indicator at all times.) This inhibition of clearing may be removed only by the ICOZ instruction. (See page 25.)

ICOZ 0002 *Remove Inhibition on Clearing the Overflow Indicator*

The ICOZ instruction removes any inhibition on clearing the Overflow Indicator set by the ICOS instruction. (See page 25.)


START
(M) → D
EXTENDED?
YES
32K: $D_{1\text{-}15}$ + 1 (MOD 2^{15}) → $D_{1\text{-}15}$ AND JA
65K: $D_{0\text{-}15}$ + 1 (MOD 2^{16}) → $D_{0\text{-}15}$ AND JA
NO
S = 0?
YES
32K: $D_{1\text{-}15}$ + 1 (MOD 2^{15}) → $D_{1\text{-}15}$ AND JA
65K: $D_{1\text{-}15}$ + 1 (MOD 2^{15}) → $D_{1\text{-}15}$
$D_{1\text{-}15}$ + 1 (MOD 2^{16}) → JA
NO
$D_{4\text{-}15}$ + 1 (MOD 2^{12}) → $D_{4\text{-}15}$
$D_{4\text{-}15}$ + 1 → JA
(D) → M
EXIT

INCAL
Micro-Flow Chart


START
(M) → D
EXTENDED?
YES
32K: $D_{25\text{-}39}$ + 1 (MOD 2^{15}) → $D_{25\text{-}39}$ AND JA
65K: $D_{24\text{-}39}$ + 1 (MOD 2^{16}) → $D_{24\text{-}39}$ AND JA
NO
S = 0?
YES
32K: $D_{25\text{-}39}$ + 1 (MOD 2^{15}) → $D_{25\text{-}39}$ AND JA
65K: $D_{25\text{-}39}$ + 1 (MOD 2^{15}) → $D_{25\text{-}39}$
$D_{25\text{-}39}$ + 1 (MOD 2^{16}) → JA
NO
$D_{28\text{-}39}$ + 1 (MOD 2^{12}) → $D_{4\text{-}15}$
$D_{28\text{-}39}$ + 1 → JA
(D) → M
EXIT

INCAR
Micro-Flow Chart

INCAL 0021 *Increase Address Field in Memory*

INCAR 2021

The INCA instruction transfers the contents of the specified memory location to the D Register. One is added to the left or right V-field of the contents of D, depending on whether this instruction is an INCAL or an INCAR. The sum is modulo memory size and is placed in JA. However, only 12 or 15 bits are replaced in the specified half of the D Register, depending on the setting of the S-bit in that half of D, unless the INCA instruction is extended by an EXT instruction (see pages 88-92).

The modified word replaces the contents of D and the specified memory location; the correct value of the incremented V-field remains in JA.

SKC 0012 *Skip Check*

The SKC instruction checks the status registers of input-output units. The specific units, status registers within the unit, and a Comparison Quantity to be used in testing the registers are specified in the address field of the instruction. (See Input-Output Systems Manual TM-16.) If the Comparison Quantity in the address field of this instruction is greater than or equal to the contents of a specified status register, the next sequential instruction is skipped and the second instruction following the SKC is executed. Otherwise, the next sequential instruction is executed.

This instruction acts as an NOP if it is under control of an RPT or DR instruction.

SKF 2012 *Skip if No Fault*

The SKF instruction is the same as the SKC instruction, except that specified fault registers of the input-output unit are tested.

RPT 0023 *Repeat*

RPT 2023

The RPT instruction causes the next instruction or instruction pair following the RPT instruction to be performed the number of times specified in the address field of the RPT instruction. If the RPT instruction is in the left half of an instruction word, the right-half instruction in that word is performed the number of times specified. If the RPT instruction is in the right half of an instruction word, the next pair of instructions is performed the number of times specified.

The value in the address field can be any number up to 4095. If the RPT instruction is extended by an EXT instruction (see pages 88-92, the value in the address field can be large enough to express the size of memory. If the value is zero, the Repeat instruction has no effect and the otherwise repeated instructions are ignored.

If a Jump Instruction that is located in the left half of an instruction word is being performed under the RPT instruction, the number of times the instruction following the Jump instruction has been performed can be determined from the contents of the JA Register, unless the transfer of control occurs the first time the instructions controlled by the RPT are being performed. The number of times the right-half instruction has been performed may be determined by storing the contents of the JA Register, and then evaluating the expression:

$$(JA) - U + W$$

where (JA) = the contents of the JA Register
U = 4096 if the RPT instruction is not extended. If the RPT instruction is extended, U is equal to the number of memory locations in the system
W = the original number of times specified for the instructions controlled by the RPT to be performed

If an instruction in a repeat loop causes a transfer of control, the repeat loop is negated.

If an RPT instruction has a second RPT or a DR instruction in its loop, the second RPT or DR instruction negates the repeat loop established by the first RPT instruction and establishes a new loop. However, an extended RPT or DR instruction within a repeat loop causes a command fault.

RPT-Controlled Index Register Modification

The Repeat instruction cannot be index register modified; however, the instruction(s) repeated can specify index registers and their operation can be controlled by appending an N, A or S to the RPT instruction. These modifiers cause the first four bits of the address field to be set as indicated in the illustration below.

0 1	2 3	4 15	16 23
N A S	N A S		RPT

Bits 0-1 specify the index register modification mode for the instruction in the left half of the instruction word being repeated.

Bits 2-3 specify the index register modification mode for the instruction in the right half of the instruction word being repeated.

Bits 4-15 designate the number of times the instruction(s) controlled by the RPT are to be repeated.

A configuration of 00 or 01 in bits 0-1 or 2-3 calls for Normal modification, 10 (specified by an A) calls for Additive modification, 11 (specified by an S) calls for Subtractive modification. One or two of these modification codes, depending on whether the RPT is in the left or right half of an instruction word, must be appended to the RPT. (Example: RPTN, RPTSN.)

Normal modification: Normal index register modification using the C-bit and the Y-bit (see page 21).

Additive modification: The C-bit and Y-bit are ignored. The contents of the index register are used as the effective address of the instruction, then the index register is modified by adding the V-field of the instruction to the contents of the index register.

Subtractive modification: The C-bit and Y-bit are ignored. The contents of the index register are used as the effective address of the instruction, then the index register is modified by subtracting the V-field of the instruction from the contents of the index register.

The A and S index register modification modes have no effect on any Index Register Instruction except TCX. The TCX instructions act as Indexable Instructions when in the repeat-controlled index register modification mode.

DR 1312 *Double Repeat*

The DR instruction causes the next three instructions (if DR is in the left half of an instruction word) or the next four instructions (if DR is in the right half of an instruction word) to be performed the number of times, from zero to 255, specified in the address field of the DR instruction. .If the DR instruction is extended by an EXT instruction (see pages 88-92), the value in the address field can be large enough to express the size of memory. If this value is zero, the instructions under the DR will be skipped.

If a Jump Instruction that is located in the left half of an instruction word is being performed under the DR instruction, the number of times the instructions following the Jump Instruction have been performed can be determined from the contents of the JA Register, unless the transfer of control occurs the first time the instructions controlled by the DR are being performed. The number of times the instructions following the Jump Instruction have been performed can be determined by storing the contents of the JA Register, and then evaluating the expression:

$$(JA) - U + W$$

where (JA) = the contents of the JA Register
U = 4096 if the DR instruction is not extended. If the DR instruction is extended, U is equal to the number of memory locations in the system
W = the original number of times specified for the instructions controlled by the DR to be performed

SPECIAL

If an instruction in a repeat loop causes a transfer of control, the repeat loop is negated.

If a DR instruction has an RPT or a second DR instruction in its loop, the RPT or the second DR instruction negates the repeat loop established by the first DR instruction and establishes a new loop. However, an extended RPT or DR instruction (see page 88) within a repeat loop causes a command fault.

DR-Controlled Index Register Modification

The DR instruction cannot be index register modified; however, the index registers designated by the instructions under a DR instruction can be index register modified as indicated by the Normal, Additive and Subtractive modification codes which are described under the RPT instruction. These modifiers cause the first eight bits of the address field to be set as indicated in the illustration below.

0 1	2 3	4 5	6 7	6 15	16 23
N A S	N A S	N A S	N A S		DR

Bits 0-1 specify the index register modification for the instruction in the left half of the first instruction word being repeated if the DR is not in the left half of the instruction word.

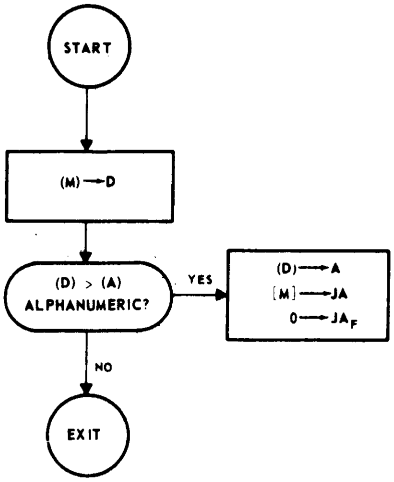
Bits 2-3 specify the index register modification mode for the instruction in the right half of the first instruction word being repeated.

Bits 4-5 designate the index register modification mode for the instruction in the left half of the second instruction word being repeated.

Bits 6-7 designate the index register modification for the instruction in the right half of the second instruction word being repeated.

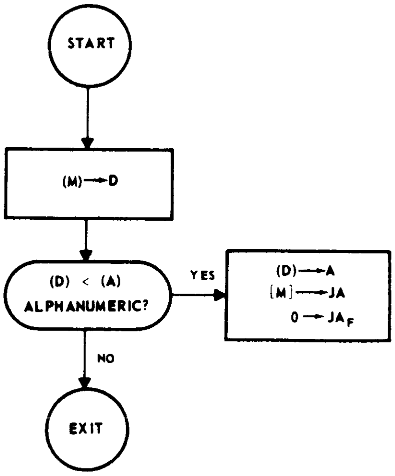
Bits 8-15 specify the number of times the instruction word(s) under the DR are to be repeated.

LWD **0033** *Larger Word*



The LWD instruction transfers the operand from a specified memory location to the D Register, then compares the contents of the D Register to the contents of the A Register bit-by-bit in the alphanumeric sense (see page 15). If the operand is larger than the word in A, it is transferred to A and its address is placed in the JA Register. The F-bit of the JA Register is set to zero, and the next sequential instruction is executed. If the operand from memory is smaller than or equal to the work in A, the next sequential instruction is executed. The operand remains in D.

SWD **2033** *Smaller Word*



The SWD instruction transfers the operand from a specified memory location to the D Register, then compares the contents of the D Register to the contents of the A Register bit-by-bit in the alphanumeric sense. If the operand from memory is smaller than the word in A, it is transferred to A, its address is placed in the JA Register, the F-bit of the JA Register is set to zero, and the next sequential instruction is executed. If the operand from memory is larger than or equal to the word in A, the next sequential instruction is executed. The operand remains in D.

EXT **1313** *Extend*

The EXT instruction extends the address field of the next instruction to include all bits necessary for the largest computer address, thus forming an extended V-field. The extended V-field, the command field of the instruction following the EXT instruction, and parts of the address field of the EXT instruction form an extended instruction (see illustrations below). An EXT instruction may be in the left or right half of an instruction word. The formats shown below assume the EXT instruction is in the left half of a word.

Bit 13 is a repeat indicator. It must be zero to indicate that the extended instruction is not a DR or an RPT; otherwise a command fault results. If the repeat indicator is one, the extended instruction must be an RPT or DR or a command fault results.

The format of an extended instruction other than an RPT or DR is shown below. Bits 14 and 15 are indirect addressing indicators. When they are both set to zero, the S, N, and V fields of the extended instruction define its effective address. (Bits 4-12 should be set to zero.)

0	1 3	4 12				16 23	24 39	40 47
S	N		0	0	0	EXT	V	COMMAND

The format of an extended RPT or DR instruction is shown below. The index register modification codes and the V and Repeat fields comprise the extended instruction. The repeat index register modification codes shown below are for a DR. The repeat index register modification codes for an RPT occupy only the first four bits of the address field of the EXT instruction. Bits 14 and 15 are indirect addressing indicators. When they are both set to zero, the V-field of the extended instruction specifies the number of times the instruction(s) in the repeat loop are to be performed. (Bits 4-12, unless they are defined, should be set to zero.)

0 7	8 12				16 23	24 39	40 47
REG.		1	0	0	EXT	V	REPEAT

When an extended instruction replaces a V-field in the D Register or in memory (INCA, TXD, TJM), transfers a V-field from D to an index register (TDX), adds or subtracts a V-field in D from an index register (ADX, SDX), or uses the V-field(s) in D for a comparison or a jump address (AIXJEG, AIXJS, AIXJ, SIXJ, SIXJES, SIXJG, AIXOL and AIXOR), the S-bit(s) in D do not determine the size of the V-field(s) in D. The V-field(s) are modulo memory size.

An extended instruction that replaces a V-field in D or in memory on a 65K system replaces all 16 bits of the address field in D; on other systems the S-bit is set to zero and a full V-field is replaced.

Instructions in a repeat loop can be extended. If an extended instruction is under a repeat-controlled index register modification mode, the index register modification mode of both the EXT and the extended instruction must be identical.

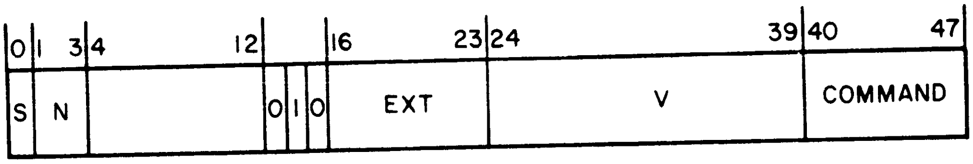
If an EXT instruction is the last instruction in a repeat loop, or an extended RPT or DR instruction is in a repeat loop, a command fault results.

Indirect Addressing

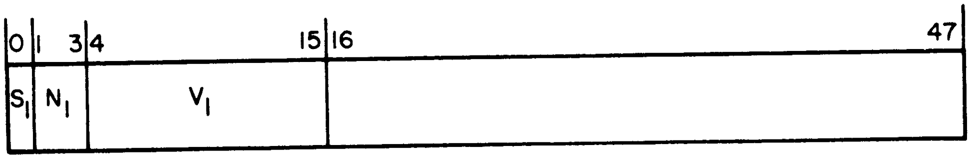
Indirect addressing is the substitution of the address field(s) of an instruction by the corresponding field(s) of a memory word in the location specified (referenced) by that instruction. If bits 14 and 15 of an extended instruction are not both set to zero, indirect addressing is indicated. There are three possibilities:

Bit 14	Bit 15	Indirect Addressing
1	0	The address specified by the address field(s) of this extended instruction is referenced. The address field(s) in the left half of the word referenced (which are in the format as described on page 15) are used in forming the effective address of the command in the original extended instruction. The command of the original extended instruction and the address field(s) used to form its effective address form an effective instruction.

If the extended instruction in the illustration below is an Indexable Instruction, the S-, N-, and V-fields of the extended instruction specify the address of the referenced word in the diagram. The Command of the original extended instruction and the S_1-, N_1-, and V_1-fields form the effective instruction. Index registers specified by each address are modified.

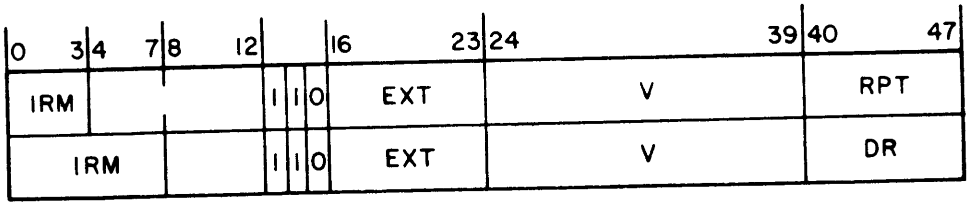


EXTENDED INSTRUCTION

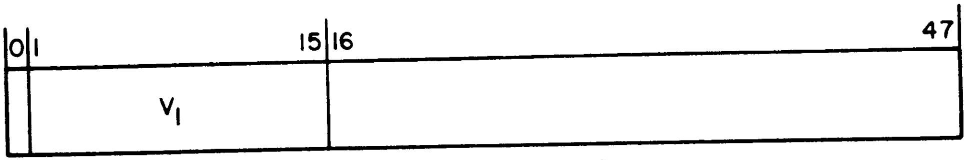


REFERENCED WORD

The V-field of the extended RPT or DR instruction in the illustration below specifies the address of the referenced word in the diagram. The original RPT or DR command, its index register modification codes (IRM) and the V_1-field form the effective instruction. (For 65K, the V_1-field includes bit zero.)



EXTENDED INSTRUCTION



REFERENCED WORD

Bit 14 Bit 15 Indirect Addressing

1 1 Same as above, except that the address field(s) that are used in forming the effective address of the original extended instruction are in the *right* half of the referenced word.

Bit 14	Bit 15	Indirect Addressing
0	1	The address specified by the address field(s) of the extended instruction is referenced.

The address field(s) of the word referenced are in the same format as the address field(s) in an extended instruction, and are used in forming a new address for the command in the original extended instruction. (Bits 4-12 of the referenced word should be set to zero.) *If the indirect addressing control bits of the referenced word are both set to zero,* the new address formed is the effective address of the original extended instruction. If the indirect addressing control bits of the referenced word are *not* both set to zero, they determine the format of a second word to be referenced for its address field(s). The second word to be referenced is in the location specified by the new address of the original extended command. This process may continue until an indefinite number of words are referenced.

In the illustration below, the S-, N-, and V-fields of the Indexable Instruction specify the address of the first referenced word in the diagram. The new address of the Command in the extended instruction is formed by the S_1-, N_1-, and V_1-fields. This new address specifies the address of the second referenced word. The effective instruction is formed by the Command in the original extended instruction and the S_2-, N_2-, and V_2-fields. Index registers specified by each address are modified.

0	1 3	4 12				16 23	24 39	40 47
S	N		0	0	1	EXT	V	COMMAND

EXTENDED INSTRUCTION

0	1 3	4 12				16 23	24 39	40 47
S_1	N_1			1	0		V_1	

FIRST REFERENCED WORD

0	1 3	4 15	16 47
S_2	N_2	V_2	

SECOND REFERENCED WORD

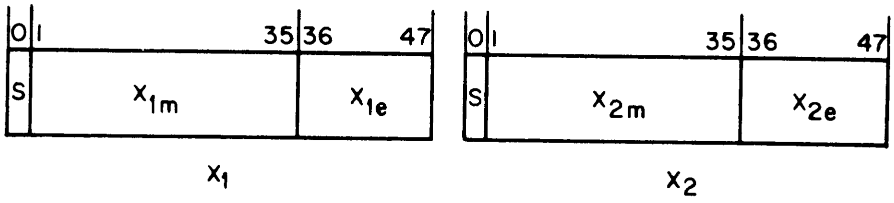
Chapter 7

DOUBLE PRECISION ARITHMETIC

Double precision arithmetic permits operating on double-length floating-point operands with mantissas of 70 bits plus a sign bit, and exponents (of the base 2) of 11 bits plus sign bit. The operations which are performed are addition, subtraction, and multiplication. The mantissas of results are accurate to 70 bits for addition and subtraction, and to 69 bits for multiplication. Results are significant to 22 decimal places for addition and subtraction or to 21 decimal places for multiplication.

OPERAND FORMAT

When in double precision mode, the format of a double-length operand, or result, X, is as follows:



where:

X_1 = major half of the operand

X_2 = minor half of the operand

X_{1m} = major half mantissa

X_{2m} = minor half mantissa

X_{1e}, X_{2e} are equal, and each represents the exponent of the operand to the base 2

S in the major half of the mantissa denotes the sign bit of the double-length operand. (The S-bit in the minor half of the mantissa is not significant.)

Double precision operands need not be in any particular memory locations, nor need the major and minor halves be held in consecutive memory locations.

Negative mantissas and negative exponents are represented in the two's complement form.

Since all arithmetic operations on the Philco 212 are performed using 48-bit registers, the double precision operations are performed by setting the computer in double precision mode, operating on the major and minor halves of operands separately, and delaying normalization until the next ENDDP is executed.

The operations which should be performed on two double-length operands, X and Y, when the computer is in double precision mode are:

$$(1)\quad X + Y = (X_1 + Y_1) + (X_2 + Y_2)$$

$$(2)\quad X - Y = (X_1 - Y_1) + (X_2 - Y_2)$$

$$(3)\quad XY = (X_1 + X_2)(Y_1 + Y_2)$$

$$= X_1Y_1 + X_1Y_2 + X_2Y_1 + X_2Y_2$$

Since the expression X_2Y_2 would contribute an accuracy beyond bit 70, it is dropped from the equation for multiplication, so that:

$$XY = X_1Y_1 + X_1Y_2 + X_2Y_1$$

Double Precision Zero

Double precision floating-point zero is represented by a major and minor half operand, each with a mantissa of 36 zero bits and an exponent with a sign bit of one followed by 11 zero bits.

DOUBLE PRECISION MODE

The SETDP instruction sets the computer in double precision mode. When in this mode, the following instructions are specially modified to use in routines for obtaining double precision results: FAQS, FSQS, FAM, FSM, FMAS, and FMMR. The operation of these instructions, together with their specific use in forming double precision results, is described below. No other arithmetic instructions should be used when in double precision mode.

After a double precision operation, an ENDDP instruction, as described below, is given. This normalizes and adjusts the halves of the result in the A and Q Registers and causes the computer to terminate double precision mode.

Correction Counter

Because partial results (major half and minor half) are formed during double precision operations, any potential effect of the minor-half operation on the major-half operation is recorded in a correction counter. This "spillover" consists of the normal carry data from one bit to the next most significant bit for addition and subtraction. For multiplication the spillover consists of the carry-out data from the summation of the minor-half partial products $(X_1Y_2 + X_2Y_1)$ and the effect of the sign bits of the minor-half partial products on the major-half partial product (X_1Y_1). If a minor-half partial product is negative, the value of all bits preceding the most significant bit must be one. Therefore, when a negative minor-half cross product is formed, a mantissa of 36 bits, all with a value of one, must be added to the major-half result.

A three-bit correction counter records any spillover. After the execution of an ENDDP instruction, the correction counter is cleared. For multiplication it is set to -2 during the execution of an FMAS instruction. Each time a bit is carried from the most significant bit of a result during a minor-half addition or subtraction, or during the summation of the minor-half partial products, the counter is incremented by one. It is also incremented by one if a positive minor-half partial product is formed. When an ENDDP instruction is executed, the mantissa of the major-half result is altered according to the setting of the correction counter as described in the table below. The result in A and Q is then normalized.

Setting of Correction Counter	Correction Made To Major Half Mantissa
-2	A mantissa of 35 one's preceding a zero is added to the major-half mantissa.
-1	A mantissa of 36 one's is added to the major-half mantissa.
0	No correction.
1	A one bit is added to bit 35 of the major-half mantissa.
2	A one bit is added to bit 34 of the major-half mantissa.

FAQS and FSQS

The FAQS and FSQS instructions are used to form the major half result ($X_1 + Y_1$ or $X_1 - Y_1$) in double precision addition and subtraction.

The FAQS and FSQS instructions transfer the contents of the Q Register to the D Register and clear the Q Register to zero. If there is a difference between the exponents in the A and the D Registers, and it is less than 71, the contents of the register with the smaller exponent are arranged for addition or subtraction (see page 30). Any bits shifted beyond bit 35 in the A Register are shifted, in order, into the Q Register, entering the Q Register at bit one. If the exponent difference is out of range, the number with the larger exponent becomes the sum or difference.

The FAQS instruction adds the contents of the D Register to the contents of the A Register; the FSQS instruction subtracts the contents of the D Register from the contents of the A Register. The sum or difference is formed in A. The contents of A are then transferred to the D Register and to the specified memory location.

The sum or difference in A is left unchanged. If mantissa overflow occurs, it is not corrected, but is recorded by a special overflow control indicator to be handled when the next ENDDP instruction is executed.

FAM and FSM

The FAM and FSM instructions are used to form the minor-half result ($X_2 + Y_2$ or $X_2 - Y_2$) in double precision addition or subtraction. The FAM instruction is used to sum the minor-half products during double precision multiplication ($X_1Y_2 + X_2Y_1$).

The FAM and FSM instructions transfer the contents of the specified memory location to the D Register and then force the sign bits of the A and D Registers to zero. If there is a difference between the exponents of the contents of the A and D Registers, the words in the registers are arranged for addition or subtraction (see page 30). The FAM instruction then adds the contents of the D Register to the contents of the A Register; the FSM instruction subtracts the contents of the D Register from the contents of the A Register. The sum or difference is formed in A. The contents of the Q Register are then added to the contents of the A Register, unless the subtrahend was arranged during an FSQS instruction that was executed under double precision mode since the last ENDDP instruction was executed. If the subtrahend was arranged, the contents of Q are subtracted from the contents of the A Register.

The result is left unchanged. If overflow occurs during any operation, it is not corrected, and the correction counter is incremented by one. The result is placed in both A and Q.

FMAS

The FMAS instruction is used to form the major-half partial product (X_1Y_1) during double precision multiplication.

The FMAS instruction operates as described in Chapter VI (see page 44), except that the sign bit of the result in the Q Register is forced to zero; otherwise, the result is left unchanged. If overflow and exponent fault occur, they are not acted on during the instruction performance, but are recorded to be handled when the next ENDDP instruction is executed.

FMMR

The FMMR instruction is used to form the minor-half cross products (X_1Y_2 and X_2Y_1) during double precision multiplication.

The FMMR instruction transfers a word from the specified memory location to the D Register and then forces the sign bit of the word in D to zero. The word in D is then multiplied by the contents of the Q Register, forming a rounded product in the A Register.

The result is left unchanged and, if it is positive, the correction counter is incremented by one; if it is negative, the sign bit is made zero. If exponent fault or overflow occurs, it is recorded to be handled when the next ENDDP instruction is executed.

ENDDP

The ENDDP instruction transfers the contents of the specified memory location to the D and A Registers, then adds the correction value as indicated by the correction register to the contents of the A Register. The contents of A and Q are adjusted in the standard manner for overflow, exponent underflow or a zero result. If necessary, the result is then normalized (see page 29). If an exponent fault exists after the mantissa is normalized, a transfer of control as described on page 30 is made.

After the execution of this instruction, double precision mode has been removed and the correction counter cleared.

DOUBLE PRECISION PROGRAMMING

Sample routines to perform double precision addition, subtraction, and multiplication are described below.

Addition and Subtraction

These routines for double precision addition and subtraction require one word of temporary storage. A routine for double precision addition is below. Because the routine for subtraction is almost identical to the one for addition, those commands which differ for subtraction are indicated in parentheses.

Command	Address	Remarks
SETDP	$	Set double precision mode
TMA	X1$	Transfer the major half of X to the A Register
TMQ	Y1$	Transfer the major half of Y to the Q Register
FAQS (FSQS)	SUMX1Y1$	Modified instruction; add (subtract) the major halves of X and Y and store the sum
TMA	X2$	Transfer the minor half of X to the A Register
FAM (FSM)	Y2$	Modified instruction; add (subtract) the minor halves of X and Y, and transfer the sum to the Q Register
ENDDP	SUMX1Y1$	Transfer the sum (difference) of the major halves to A; adjust and normalize A and Q. Clear double precision mode

Multiplication The double precision multiplication routine requires three words of temporary storage.

Command	Address	Remarks
SETDP	$	Set double precision mode
TMA	X1$	Transfer the major half of X to the A Register
TMQ	Y1$	Transfer the major half of Y to the Q Register
FMAS	X1Y1MAJ$	Modified instruction; form the unrounded product of the major halves. Transfer the major half of the result to the specified memory location
TQM	X1Y1MIN$	Transfer the minor half of the result to the specified memory location
TMQ	X1$	Transfer the major half of X to the Q Register
FMMR	Y2$	Modified instruction; multiply major half of X by the minor half of Y, and round product
TAM	X1Y2$	Transfer the product in A to the specified memory location
TMQ	Y1$	Transfer the major half of Y to the Q Register
FMMR	X2$	Modified instruction; multiply major half of Y by the minor half of X, and round product
TMQ	X1Y2$	Transfer the rounded product formed by the major half of X and the minor half of Y from memory to the Q Register
FAM	X1Y1MIN$	Modified instruction; add the minor half of the product formed by the major halves of X and Y to the product formed by the major half of Y and the minor half of X; then add the contents of the Q Register to the sum. Transfer the final result to the Q Register
ENDDP	X1Y1MAJ$	Transfer the product of the major halves of X and Y to the A Register; adjust and normalize A and Q. Clear double precision mode

APPENDICES

Appendix A

SYMBOLS USED IN LOGIC EQUATIONS AND FLOW CHARTS

Symbol	Definition
A	A Register Bits numbered 0-47
D	D Register Bits numbered 0-47
Q	Q Register Bits numbered 0-47
AQ	A and Q Registers
X	Index Register On 32K, bits numbered 1-15 On 65K, bits numbered 0-15
JA	JA Register On 32K, bits numbered 1-15 On 65K, bits numbered 0-15
M	Word in Memory
[M]	Address of Location of M
CI	Current Instruction (Assumed always to be in the left half of an instruction word)
[CI]	Address of Memory Location of CI
$[CI]_{k-n}$	Bits k-n of [CI]
NI	Next Instruction (Assumed always to be in the left half of an instruction word)
[NI]	Address of Memory Location of NI
$[NI]_{k-n}$	Bits k-n of [NI]
OVF	Overflow Indicator

Symbol	Definition
S	Bit 0 of CI
JA_F	F-Bit of JA Register
X_C	C-Bit of Index Register
X_Y	Y-Bit of Index Register
(R)	Contents of Register R
(R)'	Complement of (R)
|(R)|	Absolute Value of (R)
$(R)_a$	Original Contents of Register R
$(R)_z$	Final Contents of Register R
R_{k-n}	Bits k-n of Register R
CT	Console Typewriter
IOCU	Input-Output Control Unit
$\cdot$	Logical AND, where $1 \cdot 0 = 0$ $0 \cdot 1 = 0$ $0 \cdot 0 = 0$ $1 \cdot 1 = 1$
$\vee$	Logical Inclusive OR, where $1 \vee 0 = 1$ $0 \vee 1 = 1$ $0 \vee 0 = 0$ $1 \vee 1 = 1$
$\wedge$	Logical Exclusive OR, where $1 \wedge 0 = 1$ $0 \wedge 1 = 1$ $0 \wedge 0 = 0$ $1 \wedge 1 = 0$

Appendix B

PHILCO 212 CODE COMBINATIONS

The following table shows the Philco 212 characters and their corresponding octal codes:

Philco Character	Octal Code	Philco Character	Octal Code
0	00	-	40
1	01	J	41
2	02	K	42
3	03	L	43
4	04	M	44
5	05	N	45
6	06	O	46
7	07	P	47
8	10	Q	50
9	11	R	51
@	12	¬	52
=	13	$	53
;	14	*	54
≡	15	<	55
&	16	#	56
'	17	␣	57
+	20	Δ	60
A	21	/	61
B	22	S	62
C	23	T	63
D	24	U	64
E	25	V	65
F	26	W	66
G	27	X	67
H	30	Y	70
I	31	Z	71
n	32	|	72
.	33	,	73
)	34	(	74
%	35	>	75
?	36	:	76
"	37	e	77

Appendix C

PHILCO 212 QUATERNARY CODE

The following table lists the Philco 212 instructions in order by their quaternary codes.

00	00	HLTL	10	00	AM	20	00	HLTR	30	00	FAM
	01	JBTL		01	AMS		01	JBTR		01	FAMS
	02	ICOZ		02	CAM		02	ICOS		02	FCAM
	03	NOPL		03	CAMS		03	NOPR		03	FCAMS
	10	TIO		10	AMA		10	TTD		10	FAMA
	11	TCM		11	AMAS		11	TDC		11	FAMAS
	12	SKC		12	CAMA		12	SKF		12	FCAMA
	13	TCXZ		13	CAMAS		13	TCXS		13	FCAMAS
	20	TJML		20	AQ		20	TJMR		20	FAQ
	21	INCAL		21	AQS		21	INCAR		21	FAQS
	22	TIJL		22	CAQ		22	TIJR		22	FCAQ
	23	RPT		23	CAQS		23	RPT		23	FCAQS
	30	ETD		30	AQA		30	ETA		30	FAQA
	31	DORMS		31	AQAS		31	AWCS		31	FAQAS
	32	EI		32	CAQA		32	EIS		32	FCAQA
	33	LWD		33	CAQAS		33	SWD		33	FCAQAS
01	00	CM	11	00	SM	21	00	SLAQ	31	00	FSM
	01	TMA		01	SMS		01	SRAQ		01	FSMS
	02	TMQ		02	CSM		02	SLAQN		02	FCSM
	03	TMD		03	CSMS		03	SRAQN		03	FCSMS
	10	TAM		10	SMA		10	SLA		10	FSMA
	11	CA		11	SMAS		11	SRA		11	FSMAS
	12	TAQ		12	CSMA		12	SLAN		12	FCSMA
	13	TAD		13	CSMAS		13	SRAN		13	FCSMAS
	20	TQM		20	SQ		20	SLQ		20	FSQ
	21	TQA		21	SQS		21	SRQ		21	FSQS
	22	CQ		22	CSQ		22	SLQN		22	FCSQ
	23	TQD		23	CSQS		23	SRQN		23	FCSQS
	30	TDM		30	SQA		30	SCD		30	FSQA
	31	TDA		31	SQAS		31	SRD		31	FSQAS
	32	TDQ		32	CSQA		32	SCD		32	FCSQA
	33	CD		33	CSQAS		33	SRDN		33	FCSQAS
02	00	JMPL	12	00	MM	22	00	JMPR	32	00	FMM
	01	JAZL		01	MMS		01	JAZR		01	FMMS
	02	JNOL		02	MMR		02	JNOR		02	FMMR
	03	JOFL		03	MMRS		03	JOFR		03	FMMRS
	10	JAPL		10	MMA		10	JAPR		10	FMMA
	11	JANL		11	MMAS		11	JANR		11	FMMAS
	12	JAEQL		12	MMAR		12	JAEQR		12	FMMAR
	13	JAEDL		13	MMARS		13	JAEDR		13	FMMARS
	20	JQPL		20	MA		20	JQPR		20	FMA
	21	JQNL		21	MAS		21	JQNR		21	FMAS
	22	JQEL		22	MAR		22	JQER		22	FMAR
	23	JQOL		23	MARS		23	JQOR		23	FMARS
	30	JDPL		30	MAA		30	JDPR		30	FMAA
	31	JAGQFL		31	MAAS		31	JAGQFR		31	FMAAS
	32	JAGQL		32	MAAR		32	JAGQR		32	FMAAR
	33	JAGDL		33	MAARS		33	JAGDR		33	FMAARS
03	00	TDXL	13	00	DAQ	23	00	TDXR	33	00	FDAQ
	01	TDXLC		01	DAQS		01	TDXRC		01	FDAQS
	02	TXDL		02	DA		02	TXDR		02	FDA
	03	TXDLC		03	DAS		03	TXDRC		03	FDAS
	10	ADXL		12	DR		10	ADXR		12	SETDP
	11	SDXL		13	EXT		11	SDXR		13	ENDDP
	12	AIXJEG		20	MAD		12	AIXJS		20	FMAD
	13	SIXJG		21	MSU		13	SIXJES		21	FMSU
	20	JL		22	EA		20	JR		22	FEA
	21	TIXZ		23	ES		21	TIXS		23	FES
	22	TDXLY		30	AD		22	TDXRY		30	FAD
	23	TXDLY		31	SD		23	TXDRY		31	FSD
	30	AIXJ		32	TYXZ		30	AIXJ			
	31	SIXJ		33	TYXS		31	SIXJ			
	32	AIXOL					32	AIXOR			
	33	SIXOL					33	SIXOR			

Appendix D

PARITY ERRORS AND HALT INSTRUCTIONS

The following two tables describe the actions which occur upon detection of an error condition or a halt instruction. Table I presents the internal interrupt operation, Table II the stop operation. For interrupt (Table I) the Auto-Control must be in the system and not inhibited. Table II applies when Auto-Control is not in the system or is inhibited.

Letters indicating the error condition or halt instruction are displayed by a status indicator on the Operator's Console. If more than one condition described below exists at the same time, the letters indicating each are superimposed on the status indicator. Receipt of an interrupt signal extinguishes the status indicator.

Upon detection of any of the error conditions or halt instructions listed in the following tables, any input-output order being performed by a device is not interrupted, any type-out order to the Console Typewriter is completed, and any data in the Console Typewriter Buffer is typed out.

TABLE I - INTERRUPT CONDITIONS

Status	Status Indicator	Central Processor Actions Before Permitting Interrupt*	Conditions Required for Notifying Auto-Control	Normal Return Address from Interrupt Routine
Instruction Parity Error (instruction about to be performed contains a parity error)	IPE	Halts before executing the faulty instruction; awaits interrupt. Indicates the error. Ends repeat, double precision and extend modes upon signal.	Auto-Control is not cycling for external reasons and computer reaches a point at which interrupt is permitted.	Address of faulty instruction.
Operand Parity Error (Last instruction processed by the Arithmetic Unit had a memory operand with a parity error)	OPE	Halts after faulty operand has been processed in Arithmetic Unit (if an extended instruction is being processed in the Index Unit, the processing is completed before the Central Processor halts); does not store results of operation. Ignores any exponent fault. Halts before executing next instruction; awaits interrupt. Indicates the error. Ends repeat, double precision modes upon receipt of interrupt signal.	Auto-Control is notified as soon as the error is detected, even if cycling.	Return address cannot be determined. It is usually one following the address of the instruction following the instruction with a faulty operand.

* If an interrupt is requested while a previous interrupt is being processed, further interrupt is inhibited until a TIO instruction to the Auto-Control Unit and a JL or JR instruction have been executed, in that order.

TABLE I - INTERRUPT CONDITIONS (Continued)

Status	Status Indicator	Central Processor Actions Before Permitting Interrupt*	Conditions Required for Notifying Auto-Control	Normal Return Address from Interrupt Routine
Store Parity Error (Parity error in data stored in memory by a store instruction)	SPE	Halts after detecting error; awaits interrupt. Indicates the error. Ends repeat double precision, and extend modes upon receipt of interrupt signal.	Memory Control is responsible for notifying Auto-Control.	Address of next instruction Central Processor would have executed if error had not occurred.
Input-Output Parity Error (Parity error in data transferred to or from memory by an input-output order)	IOPE	Central Processor not notified of error. Interrupt not accepted until repeat, double precision and extend modes are completed.	Memory Control is responsible for notifying Auto-Control.	Address of next instruction Central Processor would have executed if interrupt had not occurred.
Indirect-Addressing Parity Error (Parity error in word referenced during indirect addressing process)	OPE IPE	Halts and terminates extend mode when error is detected. Completes indexing and/or index register modification for the address formed by each word referenced during the indirect addressing process, except the one in which the error was discovered. Indicates zero.	Auto-Control notified as soon as error is detected.	Return address cannot be determined. It is usually the address of the extended instruction or one word higher.

* If an interrupt is requested while a previous interrupt is being processed, further interrupt is inhibited until a TIO instruction to the Auto-Control Unit and a JL or JR instruction have been executed, in that order.

TABLE I - INTERRUPT CONDITIONS (Continued)

Status	Status Indicator	Central Processor Actions Before Permitting Interrupt*	Conditions Required for Notifying Auto-Control	Normal Return Address from Interrupt Routine
Command Fault	CMD FLT	Halts before executing next instruction; awaits interrupt. Indicates error. Ends repeat, double precision and extend modes upon receipt of interrupt signal.	Auto-Control is not cycling for external reasons and computer reaches a point at which interrupt is permitted.	Address of faulty instruction.
HLT Instruction	HALT	If Auto-Control is cycling, allows interrupt before executing Halt instruction. If Auto-Control is not cycling, executes Halt instruction, notifies Auto-Control, indicates the Halt, and awaits interrupt. Ends repeat and double precision modes upon receipt of interrupt signal. Ends extend mode if HLT instruction is not executed.	Computer reaches a point at which interrupt is permitted.	Address of the instruction following the HLT instruction if the HLT is executed; otherwise, the address of the HLT instruction.
JBT Instruction and Breakpoint Switch Set to Halt	BRK PT	Halts before executing the instruction; awaits interrupt. Indicates the condition. Ends repeat, double precision and extend modes upon receipt of interrupt signal.	Auto-Control is not cycling for external reasons and computer reaches a point at which interrupt is permitted.	Address of the JBT instruction.

*If an interrupt is requested while a previous interrupt is being processed, further interrupt is inhibited until a TIO instruction to the Auto-Control Unit and a JL or JR instruction have been executed, in that order.

TABLE II - NO INTERRUPT

Status	Status Indicator	Central Processor Actions*	
		Detection of Error	Advance Bar Pressed
Instruction Parity Error (Instruction about to be performed contains a parity error)	IPE	Halts before executing faulty instruction; awaits operator action. Indicates error.	No action.
Operand Parity Error (Last instruction processed by the Arithmetic Unit had a memory operand with a parity error)	OPE	Halts after processing faulty operand in the Arithmetic Unit; does not store results of operation. Ignores any exponent fault. Indicates error.	Executes next instruction. Does not end repeat, double precision, or extend modes. Extinguishes indicator. †
Store Parity Error (Parity error in data stored in memory by a store instruction)	SPE	Halts after detecting error; awaits operator action. Indicates error.	Executes next instruction. Does not end repeat, double precision, or extend modes. Extinguishes indicator.†
Input-Output Parity Error (Parity error in data transferred to or from memory by an input-output order)	IOPE	Halts after detecting error; awaits operator intervention.	Executes next instruction. Does not end repeat, double precision, or extend modes. Extinguishes indicator.†

* When one of the conditions listed in this Table causes the Central Processor to halt and the Jump Switch on the Operators Console is pressed, control is transferred to the address specified by the Console Address Register without changing the contents of JA, and the status indicator on the console is extinguished.

† All operations proceed as normal until the memory bank in which this error occurred is accessed. The Central Processor then halts and cannot proceed until the fault is manually removed by the operator.

TABLE II - NO INTERRUPT (Continued)

Status	Status Indicator	Central Processor Actions*	
		Detection of Error	Advance Bar Pressed
Indirect-Addressing Parity Error (Parity error in word referenced during indirect addressing process)	OPE IPE	Halts after executing previous instruction and clears EXT controls. Completes indexing and/or index register modification for the address formed by each word referenced during the indirect addressing process, except the one in which the error was discovered. Indicates error.	No action.
Command Fault	CMD FLT	Halts before executing faulty instruction; awaits operator action. Indicates error.	No action.
HLT Instruction	HALT	Executes HLT instruction and indicates the halt.	Executes next instruction. Does not end repeat, double precision, or extend modes. Extinguishes indicator.[†]
JBT Instruction and Breakpoint Switch Set to Halt	BRK PT	Halts before executing instruction; awaits operator action. Indicates the condition.	Executes next instruction. Does not end repeat, double precision, or extend modes. Extinguishes indicator.[†]

* When one of the conditions listed in this Table causes the Central Processor to halt and the Jump Switch on the Operator's Console is pressed, control is transferred to the address specified by the Console Address Register without changing the contents of JA, and the status indicator on the console is extinguished.

[†] All operations proceed as normal until the memory bank in which this error occurred is accessed. The Central Processor then halts and cannot proceed until the fault is manually removed by the operator.

Appendix E

PROGRAM ADDRESSABLE REGISTERS

The instructions that alter the contents of a specific register are listed beneath that register in the tables below.

Index Register

ADXL
ADXR
AIXJ
AIXJEG
AIXJS
AIXOL
AIXOR
SDXL
SDXR
SIXJ
SIXJES
SIXJEG
SIXOL
SIXOR
TDXL
TDXLC
TDXLY
TDXR
TDXRC
TDXRY
TIXS
TIXZ

C-Bit

TCXS
TCXZ
TDXLC
TDXLY
TDXRC
TDXRY
TIXS
TIXZ
TYXS
TYXZ

Y-Bit

TCXS
TCXSC
TCXZ
TDXL
TDXLC
TDXLY
TDXR
TDXRC
TDXRY
TYXS
TYXZ

D Register

Add instructions, except AD, FAD*
Subtract instructions, except SD, FSD*
Multiply instructions
Divide instructions
Transfer instructions, except TDM, TDA, TDQ, TDC, TIO
Extract instructions

AWCS
CD
DORMS
INCAL
INCAR
JAEQ
JAGQ
JAGQFL
LWD
SCD
SRD
SRDN
SWD
TXDL
TXDLC
TXDLY
TXDR
TXDRC
TXDRY

Q Register

Unrounded Multiply instructions
Divide instructions
Check orders for some input-output devices
Add and Subtract instructions in double precision mode
ENDDP if A and Q are not normalized

CQ
JQEL
JQER
JQNL
JQNR
JQOL
JQOR
JQPL
JQPR
SLAQ
SLAQN
SLQ
SLQN
SRAQ
SRAQN
SRQ
SRQN
TAQ
TDQ
TMQ

* For FAD and FSD, the floating-point word in D may have been arranged for addition or subtraction (see page 30).

A Register

Add instructions
Subtract instructions
Multiply instructions
Divide instructions
Extract instructions, except ETD

CA
ENDDP
LWD
SLA
SLAN
SLAQ
SLAQN
SRA
SRAN
SRAQ
SRAQN
SWD
TDA
TMA
TQA

JA Register

Jump instructions, except JL, JR
Floating-point arithmetic instructions causing exponent fault

INCAL
INCAR
LWD
SWD
TIJL
TIJR
TXDL
TXDR
TXDLC
TXDRC
TXDLY
TXDRY

Appendix F

INSTRUCTION RUN TIMES

The following table gives average run time, in microseconds, for each Philco 212 instruction to be processed by the Central Processor. The run time given includes all time required for instruction access, operand access and execution. All run times are accurate to within ±10% and assume normal overlap of processes and no memory conflict.

Instruction	Run Time If Following Instruction [1] Is	
	Indexed	Not Indexed
ADD and SUBTRACT		
AD, SD	0.595	0.595
Fixed-Point with operand in Q	0.795	0.795
Fixed-Point with operand in memory	1.345	1.345
FAD, FSD	1.595	1.595
Floating-Point with operand in Q	1.795	1.795
Floating-Point with operand in memory	2.345	2.345
MULTIPLY		
MAD, MSU	7.745	7.745
Fixed-Point with operand in A	6.595	6.595
Fixed-Point with operand in memory	7.145	7.145
FMAD, FMSU	6.945	6.945
Floating-Point with operand in A	5.015	5.015
Floating-Point with operand in memory	5.565	5.565
DIVIDE		
Fixed-Point	16.745	16.745
Floating-Point	12.745	12.745
CLEAR	0.570	0.475
TRANSFER		
TTD	0.575	0.475
TMA, TMQ, TMD	1.025	1.025
TCM	Indeterminate	Indeterminate
TDC	[2]	[2]
TIJL, TIJR	0.570	0.515
TIO	2.000←→19.000	2.000←→19.000
TJML, TJMR	1.580	1.580
Others	0.570	0.475

Instruction	Run Time If Following Instruction [1] Is	
	Indexed	Not Indexed
EXTRACT		
EA, ES	1.825	1.825
FEA, FES	2.505	2.505
EI	2.095	2.095
EIS	2.305	2.305
ETA, ETD	1.495	1.495
INDEX REGISTER		
TDXL, TDXR TDXLC, TDXRC TDXLY, TDXRY TIXZ, TIXS TCXZ, TCXS, TCXSC	0.570	0.495
TXDL, TXDR TXDLC, TXDRC TXDLY, TXDRY	0.765	0.765
ADXL, ADXR SDXL, SDXR	0.865	0.865
AIXJ AIXJEG AIXJS SIXJ SIXJES SIXJG	1.335 3.495	1.335 3.395
AIXOL, AIXOR SIXOL, SIXOR	1.335	1.335
JUMP [3]		
JMPL, JMPR JL, JR JNOL, JNOR JOFL, JOFR JAPL, JAPR JANL, JANR JDPL, JDPR JAZL, JAZR JBTL, JBTR	0.570 2.510	0.475 2.410
JAEDL, JAEDR	0.570 2.610	0.520 2.510
JAEQL, JAEQR	0.725 2.805	0.725 2.705
JAGDL, JAGDR	0.655 2.735	0.655 2.635

Instruction	Run Time If Following Instruction① Is	
	Indexed	Not Indexed
JUMP (Continued)		
JAGQL, JAGQR	1.115 3.235	1.115 3.135
JAGQFL, JAGQFR	0.768 2.825	0.728 2.725
JQPL, JQPR JQNL, JQNR JQEL, JQER JQOL, JQOR	0.570 2.530	0.495 2.430
LOGIC	1.345	1.345
SHIFT		
First Cycle	0.570	0.495
Each additional cycle	④	④

Instruction	Run Time If Following Instruction① Is	
	Indexed	Not Indexed
SPECIAL		
RPT, DR	0.570	0.430
SETDP ICOS, ICOZ NOPL, NOPR	0.570	0.495
HLTL, HLTR EXT	0.000	0.000
INCAL, INCAR	1.385	1.385
LWD, SWD	1.595	1.595
ENDDP	1.450	1.450
SKC, SKF	1.100←→8.500	1.100←→8.500

①If an instruction is followed by an EXT instruction, the instruction extended by the EXT instruction should be examined for indexing. The run time of an instruction followed by an EXT instruction is then determined as follows:

For an extended instruction that does not specify indirect addressing, the run time of the instruction that it follows should be compared with 0.810 if the extended instruction specifies indexing or with 0.670 if the extended instruction does not specify indexing. The larger of the two values is the run time of the instruction followed by an EXT instruction.

For an extended instruction that specifies indirect addressing, the product obtained by multiplying the number of words to be referenced during the indirect addressing process by 1.370 microseconds should be calculated. If the product is greater than the run time given in the table for the instruction followed by the EXT, the product is the run time of that instruction; otherwise, the run time given in the table is the run time for the instruction.

② The time for this instruction is 30 microseconds if the Console Typewriter Buffer is not full. If the Console Typewriter Buffer is full, execution time for this instruction is 64.5 milliseconds.

③The first value given for an instruction that may cause a transfer of control is the time required if control is not transferred; the second value indicates time if control is transferred.

④A cycle is a shift of one, two or four places right or circular, or of one or two places left. The time required for a shift of more than one cycle can be determined by evaluating the formula:

$$0.495 + 0.210 (n - 1)$$

where n = number of cycles in the shift.

INDEX

INDEX (Continued)

INDEX (Continued)

INDEX (Continued)