Litton MONROE

J Engel.

ADVANCED PROGRAMMING reference manual

for Monroe 1800 models

Monroe, The Calculator Company

CONTENTS

CONTENTS (Cont.)

ILLUSTRATIONS

INTRODUCTION

The Monroe 1800 series calculators provide the basis of a calculator system that is capable of meeting most financial, scientific, engineering, and statistical requirements. The basic calculator consists of the keyboard, a memory capacity of 74 data registers and 512 program steps, a printer, and an integral magnetic card device. Both the memory capacity and the input/output capability can be expanded to accommodate a wide variety of applications.

In addition to its operation as a standard calculator, an 1800 series calculator contains many features that give it computer capabilities:

- Programmability. Programs can be stored in the calculator's memory for repeated execution. Many programs may be in memory simultaneously, depending on the size of the memory and the number of program steps involved.
- Reliability. The calculator has a standby mode in which the machine is inoperable but power is not removed from the memory and, therefore, memory contents are not destroyed.
- Versatility. All calculator functions represented on the keyboard may be programmed and stored in memory. Additionally, machine instructions (macro-instructions), which are not represented on the keyboard, are available for programming operations involving more complex decision-making features than are available from the keyboard; data and command manipulation; input/output processing; and logical (AND, OR, and Exclusive OR) operations.

RELATED DOCUMENTATION

This manual is addressed to experienced programmers. Some familiarity with computer concepts such as architecture, operating systems and/or compilers is assumed. Keyboard operations are fully described in the Operating Instructions manuals for the Model 1800 Series Calculators; basic programming techniques are discussed in the corresponding Programming Reference manuals. Furthermore, for beginning programmers, elementary programming concepts are explained in the Fundamentals of Programming manual.

HARDWARE FEATURES

The 1800 series calculators have many features and operating characteristics that equal or excel those of a mini-computer:

- Keyboard. More than 50 data entry, control, and function instructions are available from the keyboard.
- Instruction repertoire. In addition to the keyboard instructions, approximately 120 macro-instructions can be stored in memory from the keyboard or entered via peripherals.
- Memory. User-dedicated memory provides 74 data registers and 512 program steps in the basic configuration. Data registers are expandable in increments of 64 to a maximum of 522; program memory is expandable in increments of 512 to a maximum of 4096.
- Program control. Manual setting of the FLAG key and SENSE switch provides operator controlled conditional branching and looping within a program.
- Addressing modes. The calculator memory is byte addressable and register addressable; various types of indirect register addressing and symbolic program addressing are provided.
- Indexing. An index register is available for data and control manipulation.
- Input/output. Byte-serial input/output is accomplished at rates up to 500 bytes per second. Both input/output interrupt and peripheral polling may be implemented. The standard calculator provides magnetic card input and output, keyboard input, and printer output through devices that are integral parts of the calculator. In addition, many peripheral devices may be interfaced with the calculator.

SOFTWARE FEATURES

The 1800 series calculators allow storage of many programs in memory at the same time and the selective execution of those programs. Since the calculator has a standby mode, the programs may be left there indefinitely. Other programming capabilities of the calculator include the ability to perform both conditional and unconditional branching, the ability to test data and make decisions, the capability of performing subroutines, the ability to process interrupts, and the availability of a large number of instructions beyond those represented on the keyboard, such as manipulating data, commands, or addresses and performing logical OR, AND, and Exclusive OR operations.

Additionally, many programs written for specific applications are available from Monroe's program library through your local Monroe Sales Representative. The library programs include a description of the application and complete operating instructions.

PERIPHERAL DEVICES

Numerous peripheral devices may be attached to an 1800 series calculator. Each device contains an interface, which converts the peripheral signals to those required by the calculator. This interface is not part of the calculator itself. The calculator may be interfaced with a number of different devices, such as a card reader, XY plotter, I/O writer, etc. Details concerning specific devices are available from your Monroe Sales Representative.

I. CALCULATOR ORGANIZATION

Three primary elements – input/output, executive, and random access memory (RAM) are required for calculator processing (figure 1-1).

All calculator functions are controlled by the executive element. Information entered at the calculator keyboard is acted upon by the executive whether it is to be stored in RAM memory for later use, or channeled to an output device. Internal calculations are accomplished by securing data from either an input device or RAM memory, processing it under executive control, and directing the result back to RAM memory or to an output device.

INPUT/OUTPUT

The 1800 series calculators provide for input through the keyboard and magnetic card device and for output through the printer and magnetic card device. The functions of these devices are discussed in the following paragraphs. In addition to the integral input/output devices, the 1800 series calculators can accommodate a number of peripheral devices. The operational features and programming considerations for peripheral devices are described in the manuals for those peripherals.

KEYBOARD

When the machine is used strictly as a calculator, information enters the executive element through the keyboard. The functions are performed, and the results are printed.

When the machine's programming capabilities are used, information is entered from the keyboard either in the form of program instructions or data. It is processed by the executive element and stored in the memory element. Then, when the program is executed, the instructions and data are passed from the memory element to the executive element, where they are processed, and the results are either stored in memory or transferred to an output device. Any keyboard operation may be processed by the executive and stored into program memory. Additional programmable functions are available, as well as about 120 macro-instructions.

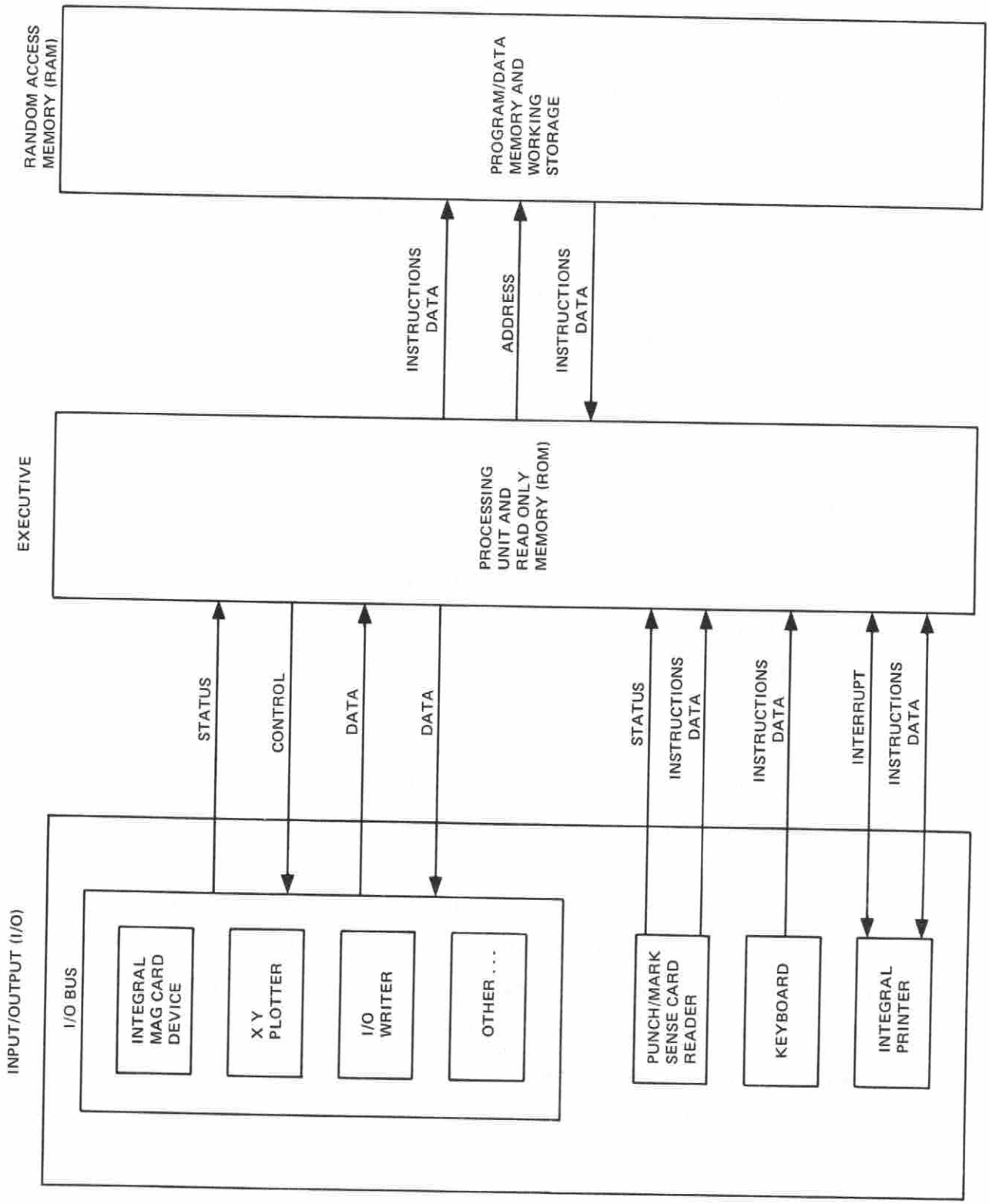


Figure 1-1. Calculator Organization

MAGNETIC CARD DEVICE

The integral magnetic card device provides storage capability for both programs and data. Information may be passed from a magnetic card, by means of the executive, into memory, or vice versa. The operation of this device requires no special programming considerations.

PRINTER

The integral printer in the 1800 series calculators provides hard-copy records of calculations, stored programs, and data as described in the Operating Instructions and Programming Reference manuals for each model.

MEMORY

The basic 1800 series programmable calculator has a memory capacity of 512 program steps and 64 main data registers (in addition to the 10 scratch pad registers). The calculator memory is segmented into pages, columns, registers, and bytes (figure 1-2). The calculator memory has four pages (numbered 0 through 3), and each page has sixteen columns numbered in octal, as follows:

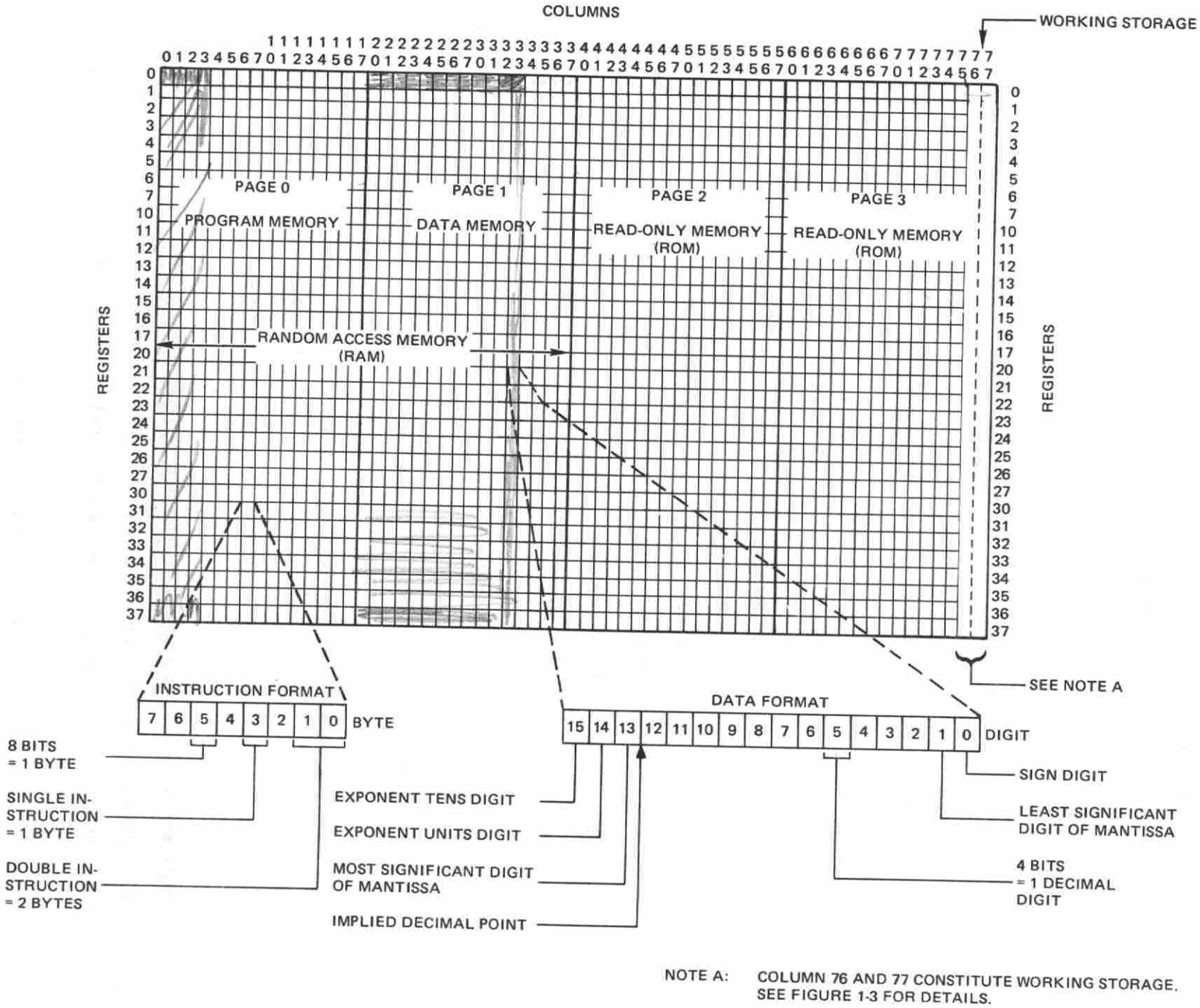
Page	contains	Columns
0		00 through 17
1		20 through 37
2		40 through 57
3		60 through 77

Columns 76 and 77 are called "working storage"; these columns are described later in this section.

Each column contains thirty-two registers (numbered, in octal, 0 through 37), and each register contains eight bytes. Bytes are numbered 0 through 7, and each byte consists of eight bits. Single instructions occupy one byte or one program step; double instructions occupy two bytes (two steps).

Page 0 is referred to as program memory and is the area in which programs are stored. Page 1 is referred to as main data memory. Pages 2 and 3 consist of working storage and read-only memory (ROM) which contains permanently-stored programs that perform the keyboard functions.

The calculator executes both keyboard instructions and macro-instructions: the codes representing keyboard instructions begin with a 0 or 1, while those for macro-instructions begin with a 2 or 3.



NOTE A: COLUMN 76 AND 77 CONSTITUTE WORKING STORAGE. SEE FIGURE 1-3 FOR DETAILS.

Figure 1-2. Memory Configuration

Macro-instructions are executed directly by the calculator hardware, whereas keyboard instructions are performed by the macro-instruction programs stored in the ROMs.

Summary of Terms

Each page contains sixteen columns (00 through 17). 16 × 32 = 512

Each column contains thirty-two registers (00 through 37).

Each register contains eight bytes (0 through 7).

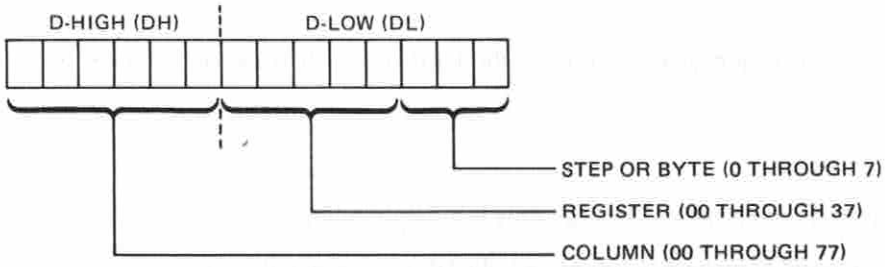
Each byte contains eight bits.

Alternate terminology:

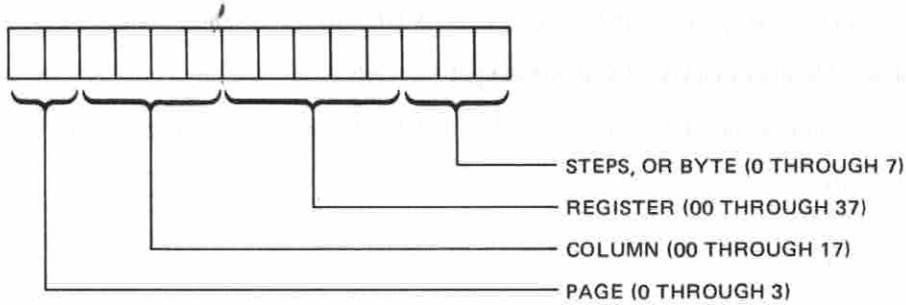
Each byte contains 2 nibbles, where 1 nibble = 4 bits = 1 digit.

MEMORY ADDRESSING

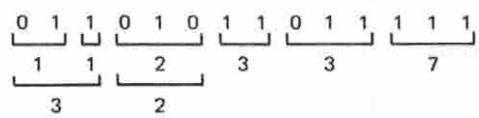
Fourteen octal digits are used to specify an address:



Page and column designations may be considered separately. In that case the page is specified as 0 through 3 and the columns as 00 through 17 (that is, each page has sixteen columns):



Thus, the address:



specifies, in octal, column 32, register 33, step 7, which is equivalent to page 1 column 12, register 33, step 7. (See, also, figure 1-5.)

INFORMATION FORMAT

Information stored in memory is treated as instructions or data. A single instruction (or step) occupies one byte; eight single instructions (or steps) can be stored in one register (figure 1-2). A double (two-step) instruction, any macro-instruction beginning with a 2, occupies two bytes.

Data is stored as one data item per register. Each register can hold 13 decimal digits together with a 2-digit exponent and a sign digit, which stores the sign for both the exponent and the mantissa (figure 1-2).

WORKING STORAGE

Columns 76 and 77 of memory are used for working storage. Most of column 77 is used by the ROM programs and is not available for general use. However, certain registers are available for use in general programs; these registers are identified in figure 1-3 and discussed in the following paragraphs.

Scratch Pad Registers

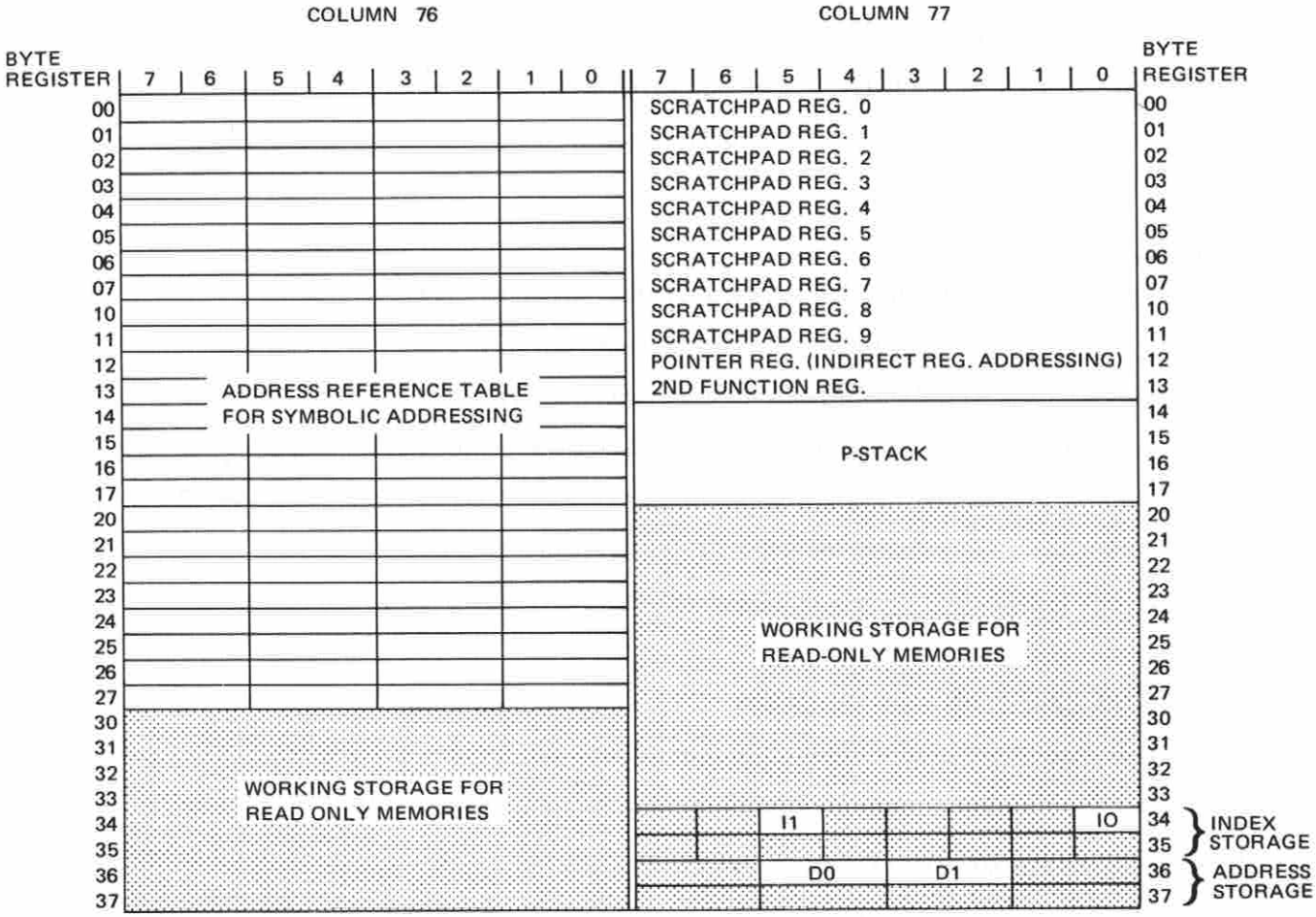
The first ten registers of column 77 (designated registers 0 through 9) are used as the scratch pad memory.

P-Stack

When any Branch instruction is executed, the address of the instruction following the Branch is stored in the P-stack. Then, when a Recall P instruction is executed, the last address stored in the P-stack is transferred into the program counter, thus defining the address of the next instruction to be executed. The P-stack is a "last-in first-out" storage area; that is, the last address stored in the P-stack is the first one to be recalled. Each address in the P-stack is referred to as a P-level. Six P-levels are always available to the general user; therefore, subroutines can always be nested at least six levels deep. The remaining portion of the P-stack is reserved for use by the ROM programs. All sixteen P-levels are available when programming strictly in macro-instructions.

Index Storage

The index register (described under PROGRAMMABLE REGISTERS) is an eight-bit register that may be used for manipulating either data addresses or instructions; performing logical AND, OR, and Exclusive OR operations; and performing bit level tests. The index register is also used to input or output information on the I/O bus. The contents of the index register (designated IX) can be stored in either of two index storage registers, which are called Index 0 (designated I0) and Index 1 (designated I1). The contents of IX



NOTE: THE SHADED AREA IS ALLOCATED AS WORKING STORAGE FOR READ-ONLY MEMORIES.

Figure 1-3. Working Storage: Columns 76 and 77

and I0 may be compared using the JINO command, described in the CONDITIONAL JUMP INSTRUCTIONS portion of Section II.

D-Storage

The address register (designated DC and described under PROGRAMMABLE REGISTERS) contains a 14-bit pointer used to address any byte or register throughout memory. The contents of DC can be stored in either of two address storage registers called D0 and D1. D0 and D1 are 16-bit registers. The address from DC is stored in bits 0 through 13. The contents of DC and D0 may be compared using the JDEO command described in Section II under CONDITIONAL JUMP INSTRUCTIONS.

EXECUTIVE ELEMENT

The executive element (see figure 1-1) controls the transfer of information between the input/output devices and RAM memory and performs the operations necessary to execute the keyboard functions and macro-instructions. A number of registers are involved in performing the functions of the executive element. Some of these registers are considered to be "programmable"; that is, the programmer can control the flow of data to and from these registers, as well as the manipulation of the data within them. Other registers are "non-programmable"; the programmer cannot control the operations performed on the data. The following paragraphs describe both types of registers.

PROGRAMMABLE CONTROL REGISTERS

The control registers which may be programmed are the entry register, address register (DC), and the index register (IX). The primary functions of these control registers are as follows:

- The entry register is used to control manipulation of numbers.
- The address register (DC) is used to control address pointers (i.e., identify memory locations of interest during data manipulation).
- The index register (IX) is used to control or manipulate digits, instructions, or flags.

Entry Register

All input data from the keyboard and output data to the printer go through the entry register (E-register). The entry register is one of the two registers involved in any data transfer. Macro-instructions are available for storing, recalling, or exchanging the contents of the entry register with the register indicated by the contents of the address register (DC).

The E-register is also involved in arithmetic calculations. At the start of a calculation the E-register contains an operand, and at the end of the operation the E-register contains the result.

The format of the entry register is shown in figure 1-4. The E-register consists of a 13-digit mantissa, a 2-digit exponent, and sign information for both the mantissa and the exponent. The ones bit of digit position 0 is set if the mantissa is negative. The twos bit of digit position 0 is set if the exponent is negative. Unlike other data registers, the E-register is provided with an overflow digit (NG) and an underflow digit (NU).

Though details of macro-instructions affecting the E-register are discussed later, the main operations are summarized below.

Conditional jumps can be performed based on the contents of the E-register. Macro-instructions are available to perform conditional jumps based on the value of the mantissa or the exponent.

The mantissa may be shifted left or right one digit at a time. (The exponent and sign digit positions are not involved in shift operations.) When shifting to the left, the contents of NU shift into digit 1 and the contents of digit 13 shift into NG. When shifting to the right, the contents of NG shift into digit 13, and the contents of digit 1 shift into NU. The original contents of NG are lost on left shifts, and the contents of NU are lost on right shifts. Following a left shift, NU is zero. Following a right shift, NG is zero.

Storing the contents of the entry register does not affect NG and NU. Recall and exchange commands set NG and NU to zero.

Address Register

The 14 bits of the address register (DC) define a location in memory by specifying page, column, register, and byte as shown in figure 1-5. DC addresses and equivalent program step numbers (page 0) are shown in figure 1-6a; DC addresses and equivalent register numbers (page 1) are shown in figure 1-6b.

To summarize the operations involving DC (discussed in detail, later), there are macro-instructions which can be used to build an address in DC. Other macro-instructions are available to increment or decrement the address in DC by one byte or one register. It should be noted that any keyboard store, recall, exchange,

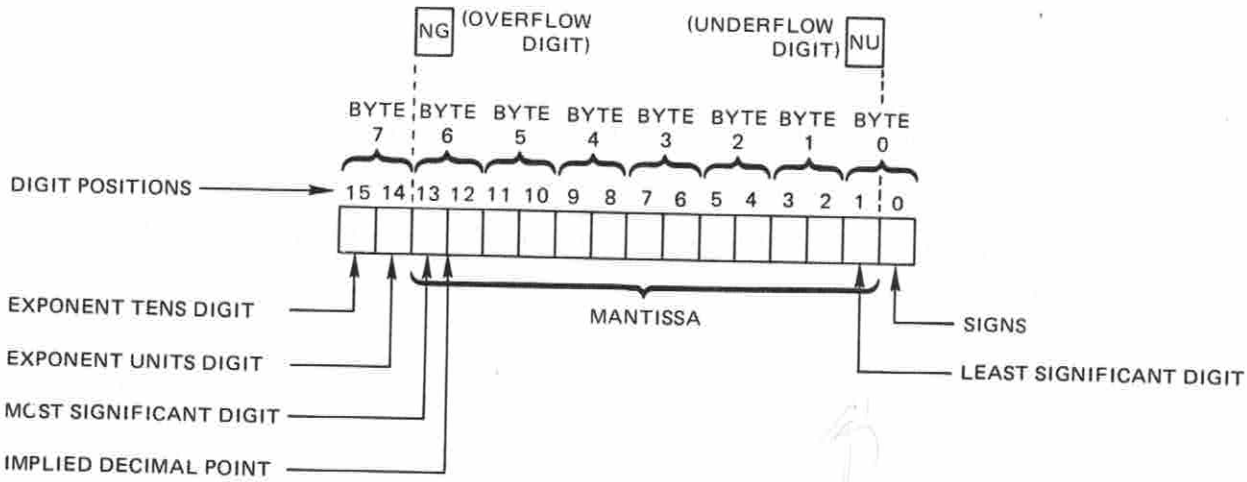


Figure 1-4. Entry Register Format

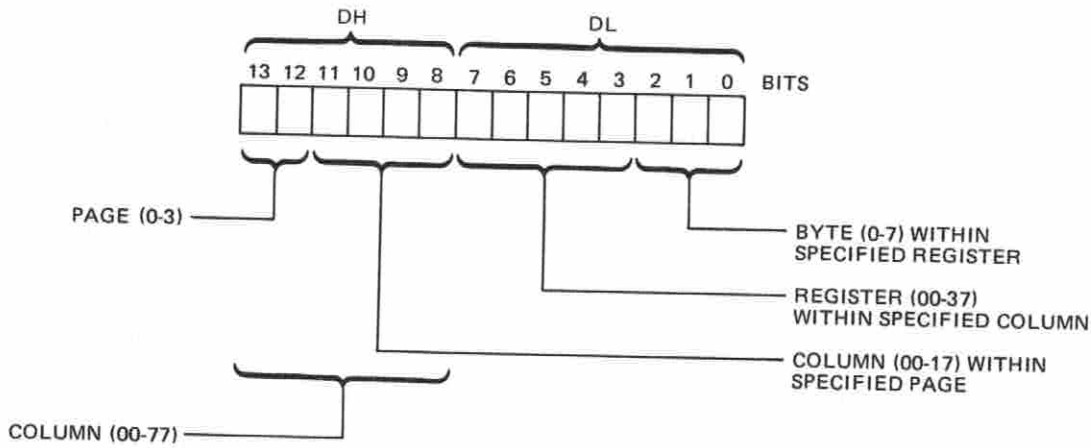


Figure 1-5. Address Register (DC)

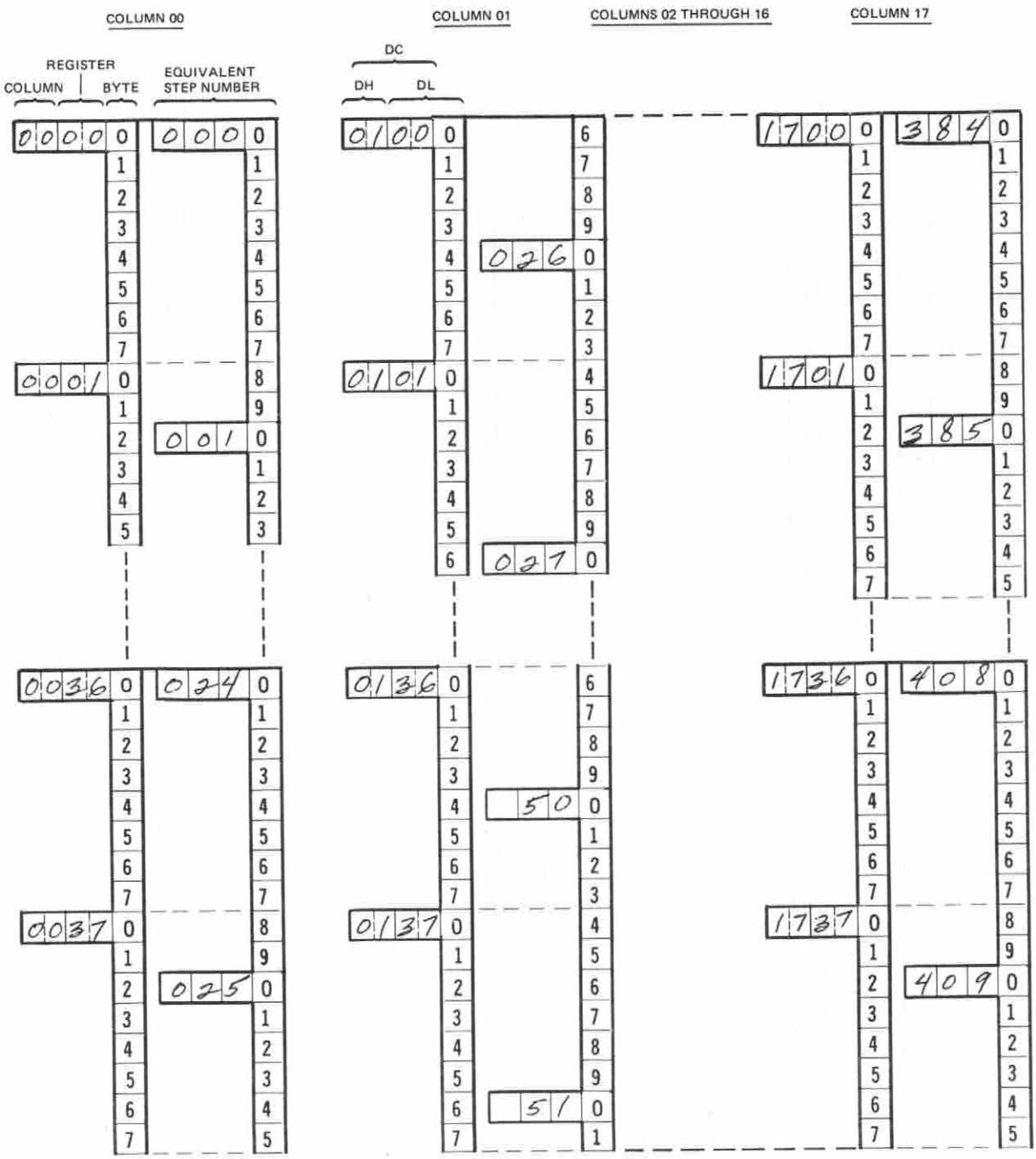


Figure 1-6a. Page 0, DC and Step Number Equivalence Diagram

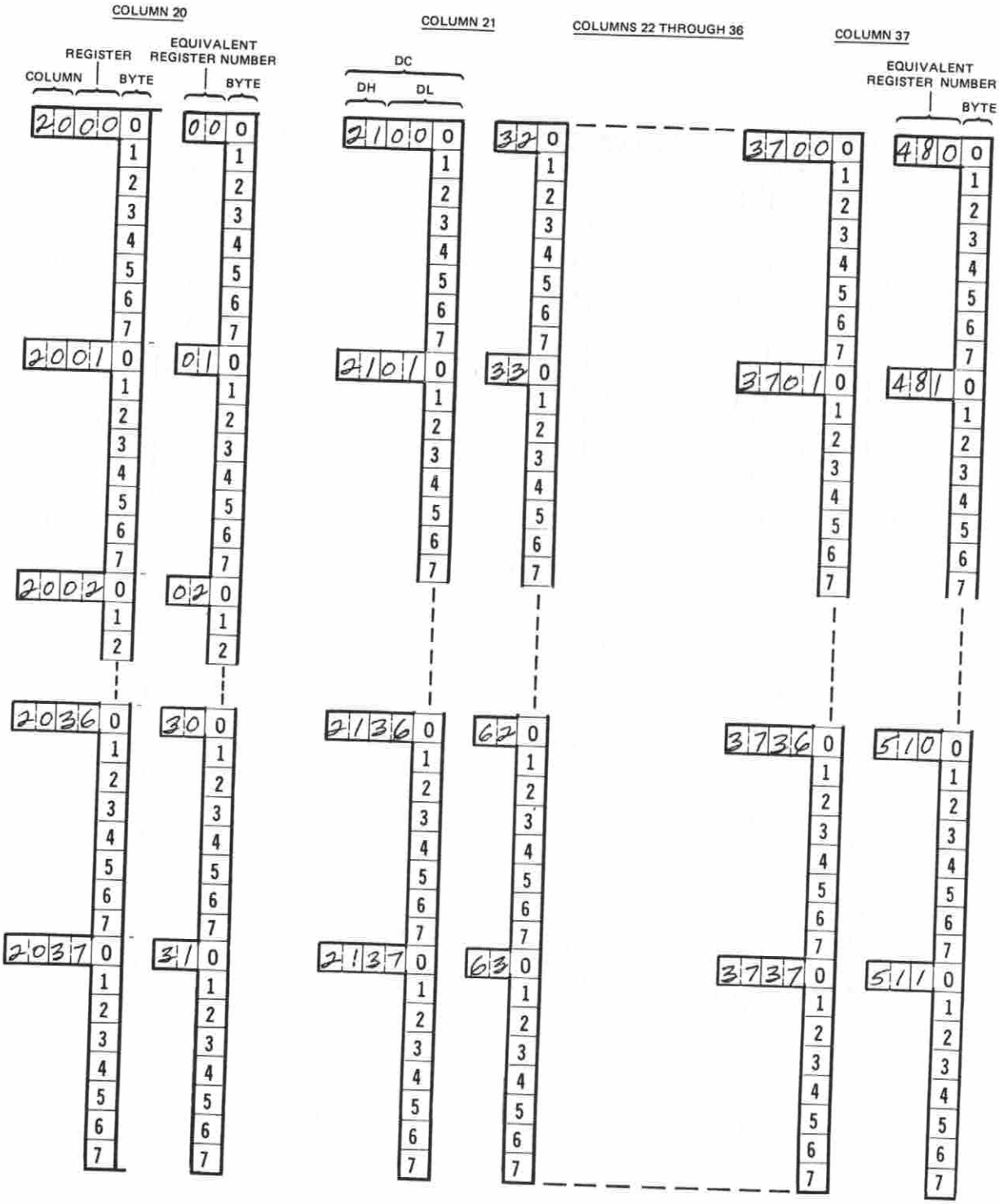


Figure 1-6b. Page 1, DC and Register Number Equivalence Diagram

or add-to-memory command (keyboard codes 120–123) leaves the address of the referenced register in DC.

Data may be transferred between DC and the P-stack. Specifically, the contents of DC may be stacked on the P-stack; DC is unchanged. Similarly, the address on the top of the P-stack may be transferred to DC; thus unstacking the P-stack one level.

Index Register

The index register (IX) is a general-purpose index and contains eight bits, as shown in figure 1-7.

The index register is divided into two areas: index high (IH) and index low (IL). During data entry, for example, each digit enters index low and then is transferred to the proper digit position in the E-register. Index high acts as a counter that specifies which digit position in the E-register is to hold the digit.

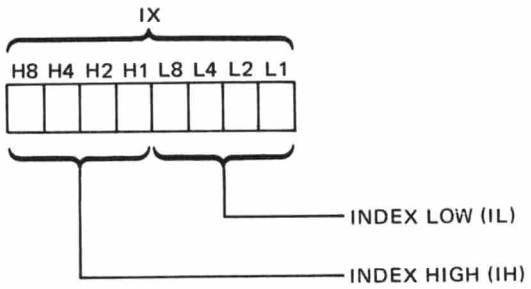


Figure 1-7. Index Register (IX)

Macro-instructions are available to load the index register; test the contents of IX and jump according to the result; perform logical operations on the contents of IX; increment and decrement the contents of IX; shift the contents left or right; and store, recall, and exchange the contents of IX with the byte specified by DC. Also, there are instructions to manipulate IX as a byte of information, or to manipulate IH and IL separately. The index register is also the vehicle for all data transfers to or from the I/O bus.

II. CALCULATOR INSTRUCTIONS

The 1800 series calculators have three types of instructions: keyboard instructions, submerged keyboard instructions,* and macro-instructions. Keyboard instructions are those that are accessible via physical keys on the calculator. Submerged keyboard instructions are identical to keyboard instructions except that they do not have corresponding keys on the calculator. Instructions are represented by the three digit octal codes 000 through 377. All keyboard instructions have a 0 or 1 as their first octal digit, and they are actually executed by ROM programs. The ROM programs are written in macro-instructions and are executed directly by the processing unit. The macro-instructions have a 2 or 3 as their first octal digit. The Keyboard instructions are defined in the applicable Operating Instructions manual. The submerged keyboard instructions and primary programming techniques are described in the applicable Programming Reference manual. The macro-instructions are defined in the following paragraphs, and are listed in Appendixes A and B. The macro-instructions are described in the following groupings: branch; jump; conditional; data manipulation (for entry, index, and address registers); and flag groups, switch groups, and the I/O bus. The description for each instruction includes the mnemonic and numeric code, as well as an illustrative example, where useful.

BRANCH INSTRUCTIONS

Branch instructions cause the calculator to copy the contents of the P-counter onto the P-stack,** to transfer control to the instruction located at address xxx (specified in the instruction), and to continue program execution from that location.

The column numbers specified in the instructions refer to the columns within a page; the first column within a page is numbered 00 for this purpose. Branching from one page to another cannot be accomplished with these instructions.

BR00,xxx Branch to Column 00 (code 200)

Branch to register-byte xxx of column 00.

Note: For example, BR00,373, (that is 200,373) would branch to column 00, register 37 (octal) byte 3. If we were on page 0 (that is, program memory) this would represent a branch to step 251 (decimal). Refer to figure 1-6a.

*The term "submerged keyboard instructions" used in this manual is equivalent to the term "non-keyboard instructions" used in the 1800 series Operating Instructions manuals and Programming Reference manuals.

**That is, branch instructions save a return address. The return address is the step after the xxx.

BR01,xxx Branch to Column 01 (code 201)

Branch to register-byte xxx of column 01.

BR02,xxx Branch to Column 02 (code 202)

Branch to register-byte xxx of column 02.

BR03,xxx Branch to Column 03 (code 203)

Branch to register-byte xxx of column 03.

BR04,xxx Branch to Column 04 (code 204)

Branch to register-byte xxx of column 04.

BR05,xxx Branch to Column 05 (code 205)

Branch to register-byte xxx of column 05.

BR06,xxx Branch to Column 06 (code 206)

Branch to register-byte xxx of column 06.

BR07,xxx Branch to Column 07 (code 207)

Branch to register-byte xxx of column 07.

BR10,xxx Branch to Column 10 (code 210)

Branch to register-byte xxx of column 10.

BR11,xxx Branch to Column 11 (code 211)

Branch to register-byte xxx of column 11.

BR12,xxx Branch to Column 12 (code 212)

Branch to register-byte xxx of column 12.

BR13,xxx Branch to Column 13 (code 213)

Branch to register-byte xxx of column 13.

or add-to-memory command (keyboard codes 120–123) leaves the address of the referenced register in DC.

Data may be transferred between DC and the P-stack. Specifically, the contents of DC may be stacked on the P-stack; DC is unchanged. Similarly, the address on the top of the P-stack may be transferred to DC; thus unstacking the P-stack one level.

Index Register

The index register (IX) is a general-purpose index and contains eight bits, as shown in figure 1-7.

The index register is divided into two areas: index high (IH) and index low (IL). During data entry, for example, each digit enters index low and then is transferred to the proper digit position in the E-register. Index high acts as a counter that specifies which digit position in the E-register is to hold the digit.

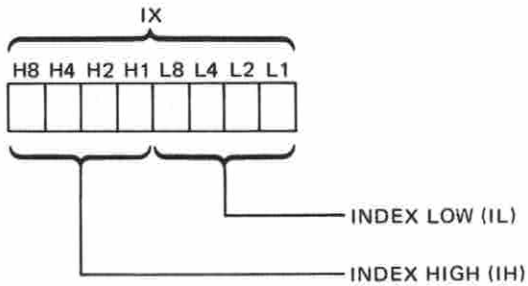


Figure 1-7. Index Register (IX)

Macro-instructions are available to load the index register; test the contents of IX and jump according to the result; perform logical operations on the contents of IX; increment and decrement the contents of IX; shift the contents left or right; and store, recall, and exchange the contents of IX with the byte specified by DC. Also, there are instructions to manipulate IX as a byte of information, or to manipulate IH and IL separately. The index register is also the vehicle for all data transfers to or from the I/O bus.

BR14,xxx Branch to Column 14 (code 214)

Branch to register-byte xxx of column 14.

BR15,xxx Branch to Column 15 (code 215)

Branch to register-byte xxx of column 15.

BR16,xxx Branch to Column 16 (code 216)

Branch to register-byte xxx of column 16.

BR17,xxx Branch to Column 17 (code 217)

Branch to register-byte xxx of column 17.

JUMP INSTRUCTIONS

Jump instructions cause the calculator to transfer control to location xxx (specified in the instruction) and to continue program execution from that location. A return address is not saved.

The column numbers specified in the instructions refer to the columns within a page; the first column within a page is numbered 00 for this purpose. Jumping from one page to another cannot be accomplished with these instructions.

JU00,xxx Jump to Column 00 (code 220)

Jump to register-byte xxx of column 00.

JU01,xxx Jump to Column 01 (code 221)

Jump to register-byte xxx of column 01.

JU02,xxx Jump to Column 02 (code 222)

Jump to register-byte xxx of column 02.

JU03,xxx Jump to Column 03 (code 223)

Jump to register-byte xxx of column 03.

JU04,xxx Jump to Column 04 (code 224)

Jump to register-byte xxx of column 04.

JU05,xxx Jump to Column 05 (code 225)

Jump to register-byte xxx of column 05.

JU06,xxx Jump to Column 06 (code 226)

Jump to register-byte xxx of column 06.

JU07,xxx Jump to Column 07 (code 227)

Jump to register-byte xxx of column 07.

JU10,xxx Jump to Column 10 (code 230)

Jump to register-byte xxx of column 10.

JU11,xxx Jump to Column 11 (code 231)

Jump to register-byte xxx of column 11.

JU12,xxx Jump to Column 12 (code 232)

Jump to register-byte xxx of column 12.

JU13,xxx Jump to Column 13 (code 233)

Jump to register-byte xxx of column 13.

JU14,xxx Jump to Column 14 (code 234)

Jump to register-byte xxx of column 14.

JU15,xxx Jump to Column 15 (code 235)

Jump to register-byte xxx of column 15.

JU16,xxx Jump to Column 16 (code 236)

Jump to register-byte xxx of column 16.

JU17,xxx Jump to Column 17 (code 237)

Jump to register-byte xxx of column 17.

CONDITIONAL JUMP INSTRUCTIONS

Conditional Jump instructions transfer control to the indicated location if the specified condition is met; otherwise, the instruction following the Jump is executed.

The Jump instruction transfers control within the column in which the Jump instruction is located except when the instruction is located in one of the last two locations of a column; in this case the jump is to the next column.

JIL1,xxx Jump If L1 Is One (code 240)

Jump to register-byte xxx if bit L1 of the index register is 1.

JIL2,xxx Jump If L2 Is One (code 241)

Jump to register-byte xxx if bit L2 of the index register is 1.

JIL4,xxx Jump If L4 Is One (code 242)

Jump to register-byte xxx if bit L4 of the index register is 1.

JIL8,xxx Jump If L8 Is One (code 243)

Jump to register-byte xxx if bit L8 of the index register is 1.

JIH1,xxx Jump If H1 Is One (code 244)

Jump to register-byte xxx if bit H1 of the index register is 1.

JIH2,xxx Jump If H2 is One (code 245)

Jump to register-byte xxx if bit H2 of the index register is 1.

JIH4,xxx Jump If H4 is One (code 246)

Jump to register-byte xxx if bit H4 of the index register is 1.

JIH8,xxx Jump If H8 Is One (code 247)

Jump to register-byte xxx if bit H8 of the index register is 1.

JIN0,xxx Jump If IX Not Equal to 10 (code 250)

Jump to register-byte xxx if the contents of the index register do not equal the contents of 10.

JDE0,xxx Jump If DC Equals D0 (code 252)

Jump to register-byte xxx if the contents of DC equal the contents of D0.

JNGZ,xxx Jump If NG Is Zero (code 253)

Jump to register-byte xxx if the contents of NG (guard digit) are 0.

JMNZ,xxx Jump If Mantissa Is Not Zero (code 254)

Jump to register-byte xxx if the mantissa portion of the value in the E-register is not zero. The exponent is ignored in this test.

JMZP,xxx Jump If Mantissa Is Zero or Positive (code 255)

Jump to register-byte xxx if the mantissa portion of the value in the E-register is equal to or greater than zero. The exponent is ignored in this test.

JXNZ,xxx Jump If Exponent Is Not Zero (code 256)

Jump to register-byte xxx if the exponent portion of the value in the E-register is not zero. The mantissa is ignored in this test.

JXZP,xxx Jump If Exponent Is Zero or Positive (code 257)

Jump to register-byte xxx if the exponent portion of the value in the E-register is equal to or greater than zero. The mantissa is ignored in this test.

JLNZ,xxx Jump If IL Is Not Zero (code 260)

Jump to register-byte xxx if the contents of IL do not equal zero.

JHNZ,xxx Jump If IH Is Not Zero (code 261)

Jump to register-byte xxx if the contents of IH do not equal zero.

DATA MANIPULATION INSTRUCTIONS

Data manipulation instructions include initialization of registers, shifting the contents of registers, storing and recalling values, and exchanging the contents of registers.

ENTRY REGISTER DATA MANIPULATION INSTRUCTIONS

- Initializing Instruction (E-Register)

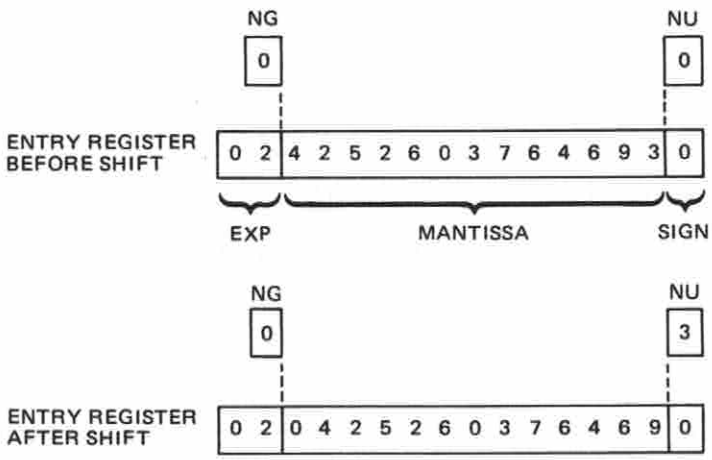
CLRE Clear E Including NG and NU (code 330)

Set the E-register (all 16 digit positions, NG and NU) to zero.

- Shift Instructions (E-Register)

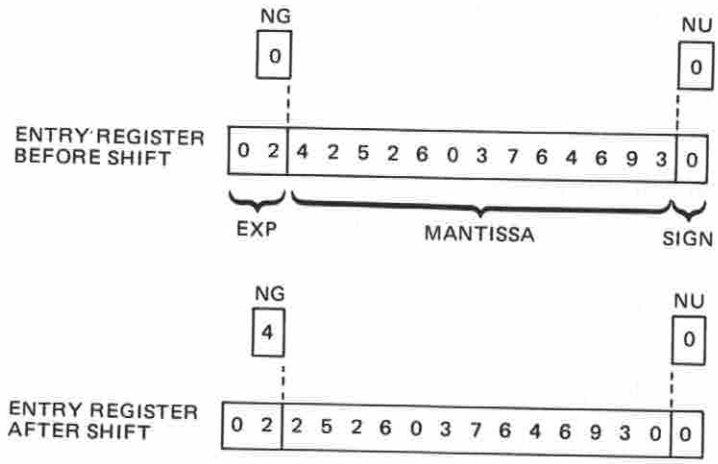
SHRE Shift Right in the E-Register (code 350)

Shift the mantissa portion of the E-register right by one digit position (four bits). The contents of NG are shifted into digit position 13, the contents of digit position 1 are shifted into NU, and NG is set to zero. The sign and exponent digits remain unchanged.



SHLE Shift Left in the E-Register (code 351)

Shift the mantissa portion of the E-register left by one digit position (four bits). The contents of NU are shifted into digit position 1, the contents of digit position 13 are shifted into NG, and NU is set to zero. The sign and exponent digits remain unchanged.



- Store, Recall and Exchange Instructions (E-Register)

STED Store E according to DC (code 331)

Store the contents of the E-register into the register location specified by the address in DC. The contents of the E-register remain unchanged.

RCED Recall E according to DC (code 332)

Copy the contents of the register specified by the address in DC into the E-register. The contents of the specified location remain unchanged. NG and NU are set to zero.

XCED Exchange E according to DC (code 333)

Exchange the contents of the E-register with the contents of the register specified by the address in DC. NG and NU are cleared.

- Increment and Decrement Instructions (E-register)

ICEX Increment Exponent (code 352)

Increment the exponent portion of the number in the E-register. This has the same effect as multiplying the contents of the E-register by 10.

DCEX Decrement Exponent (code 353)

Decrement the exponent portion of the number in the E-register. This has the same effect as dividing the contents of the E-register by 10.

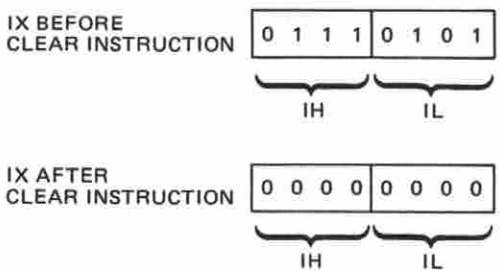
INDEX REGISTER DATA MANIPULATION INSTRUCTIONS

These instructions are used to load, increment, decrement, and store values into, or recall values from, the index register.

- Initializing Instructions (Index Register)

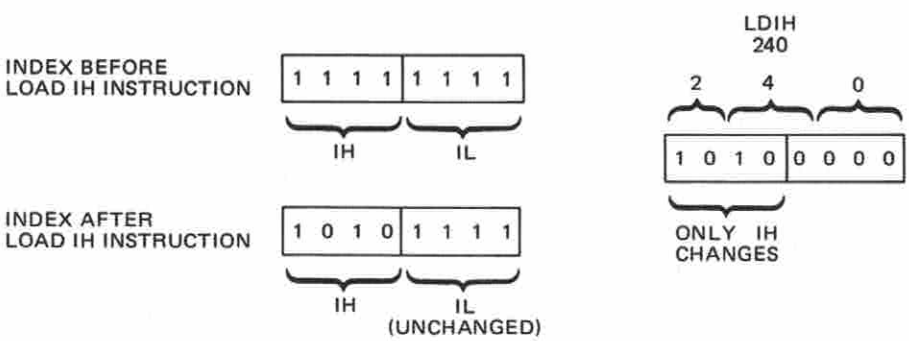
CLIX Clear Index Register (code 360)

Set the index register (IX) to zero.



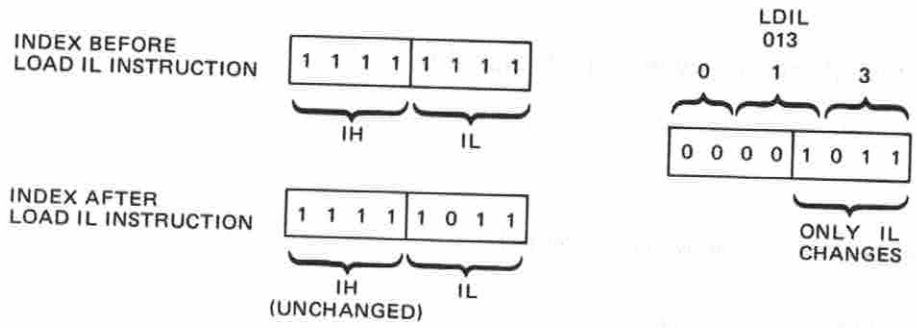
LDIH,xxx Load Index High (code 265)

Load IH with the four most significant bits of xxx; the remaining bits of xxx are ignored. Only IH is changed.



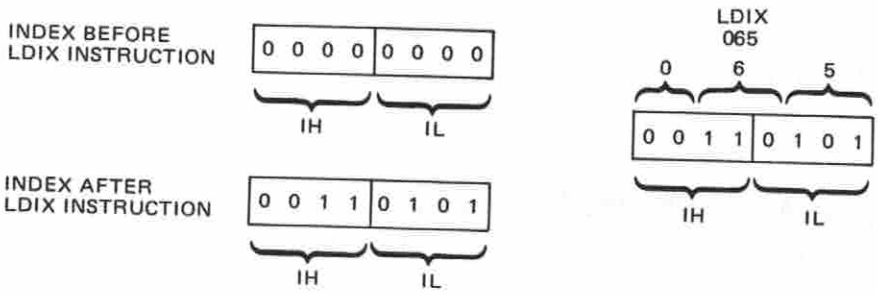
LDIL,xxx Load Index Low (code 264)

Load IL with the four least significant bits of xxx; the remaining bits of xxx are ignored. Only IL is changed.



LDIX,xxx Load Index Register (code 266)

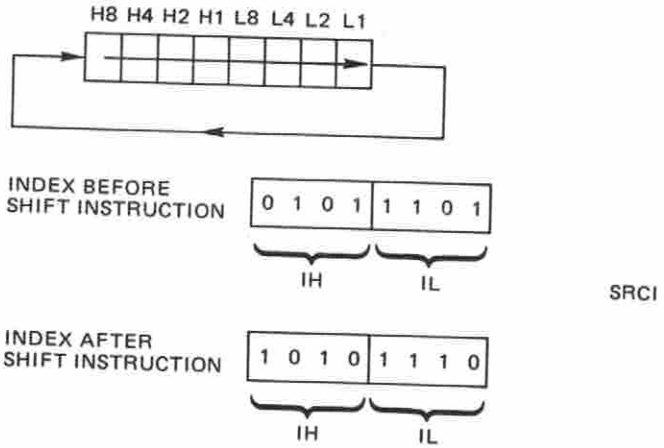
Load the index register with xxx. For example, to set index high to 3 and index low to 5, execute LDIX, 065. (Note that the following sequences are equivalent: LDIX, 065, or LDIL, 005, LDIH, 060.)



- Shift Instructions (Index Register)

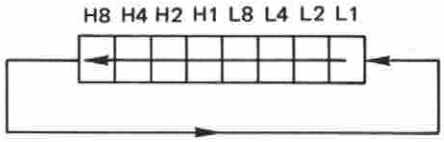
SRCI Shift Right Circular in IX (code 343)

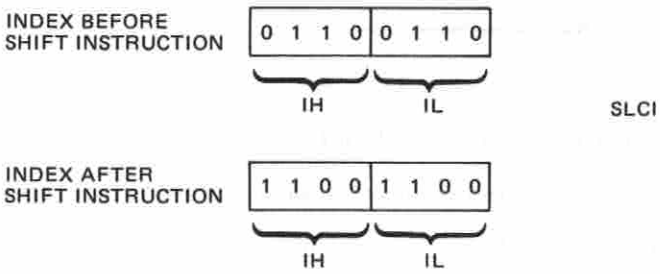
Shift the contents of the index register right by one bit. The shift is circular; that is, the initial contents of bit L1 are shifted into bit H8, so no data is lost.



SLCI Shift Left Circular in IX (code 347)

Shift the contents of the index register left by one bit. The shift is circular; that is, the initial contents of bit H8 are shifted into bit L1, so no data is lost.





- Store, Recall and Exchange Instructions (Index register)

STI0 Store IX in I0 (code 160)

Store the contents of IX in I0. The contents of IX remain unchanged.

RCI0 Recall I0 to IX (code 161)

Recall the contents of I0 to IX. The contents of I0 remain unchanged.

XCI0 Exchange IX With I0 (code 162)

Exchange the contents of IX with the contents of I0.

STI1 Store IX in I1 (code 163)

Store the contents of the IX in I1. The contents of IX remain unchanged.

RCI1 Recall I1 to IX (code 164)

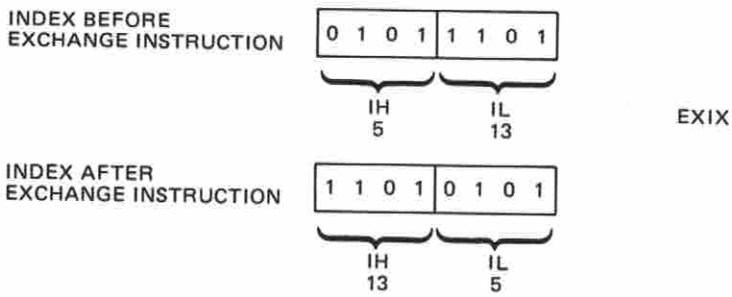
Recall the contents of I1 to IX. The contents of I1 remain unchanged.

XCI1 Exchange IX With I1 (code (165)

Exchange the contents of IX with the contents of I1.

EXIX Exchange IH With IL (code 367)

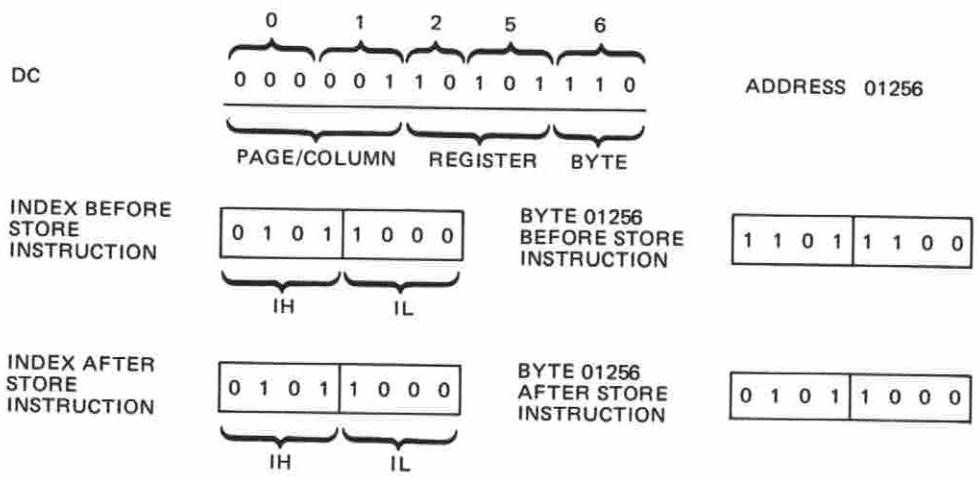
Exchange the contents of IH with the contents of IL.



STID Store IX According to DC (code 321)

Store the contents of the index register into the byte location specified by the address in DC.

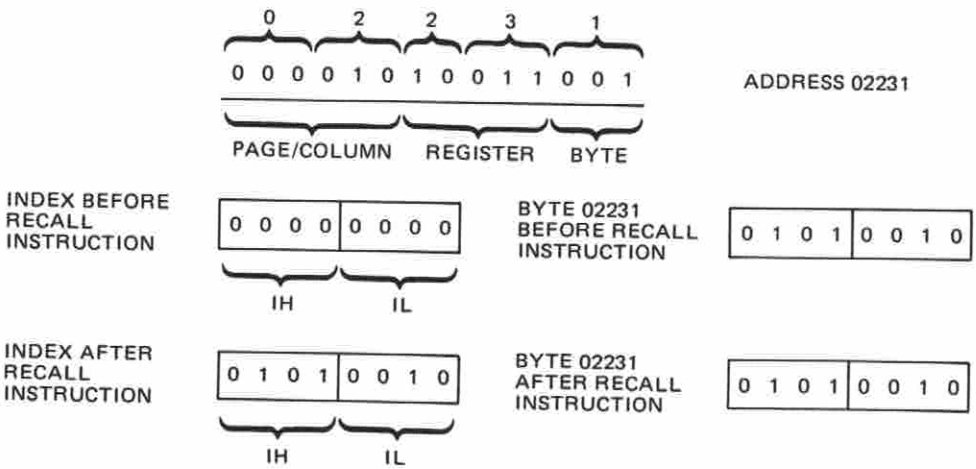
The contents of IX remain unchanged.



RCID Recall IX According to DC (code 322)

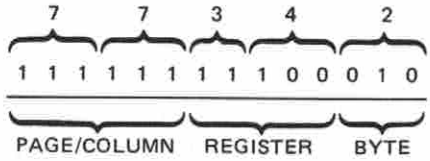
Copy the contents of the byte location specified by the address in DC into the index register.

The contents of the specified location remain unchanged.

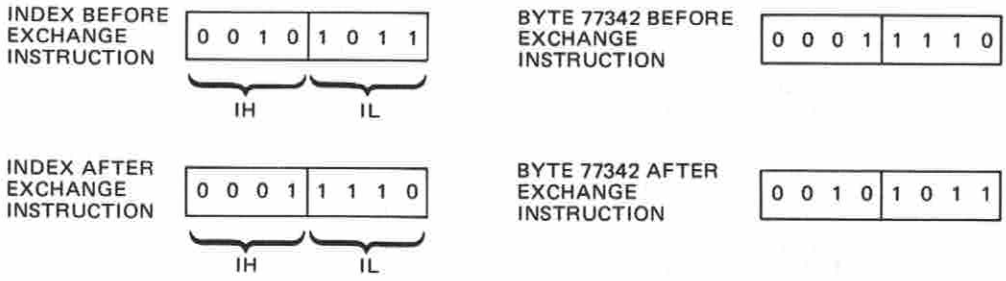


XCID Exchange IX According to DC (code 323)

Exchange the contents of the index register with the contents of the byte location specified by the address in DC.



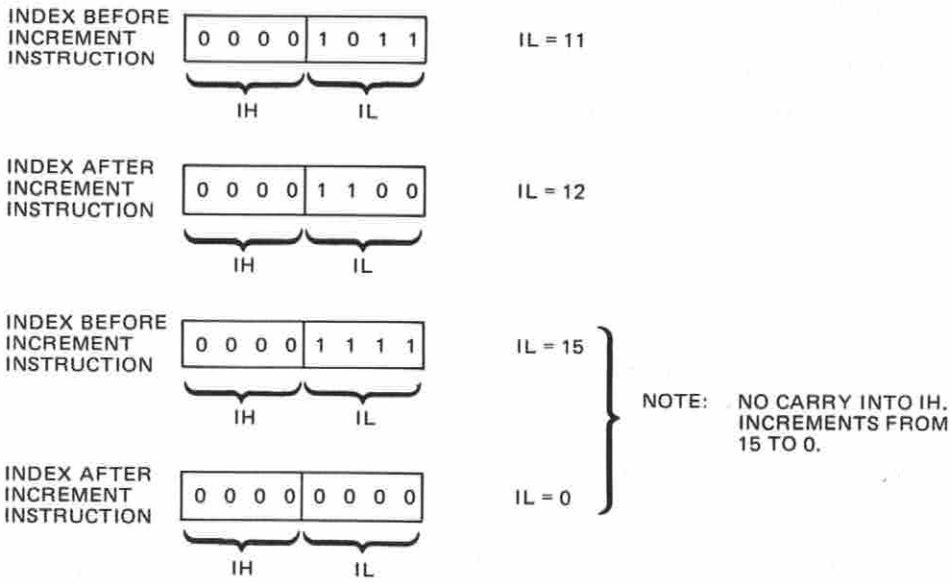
ADDRESS 77342



- Increment and Decrement Instructions (Index Register)

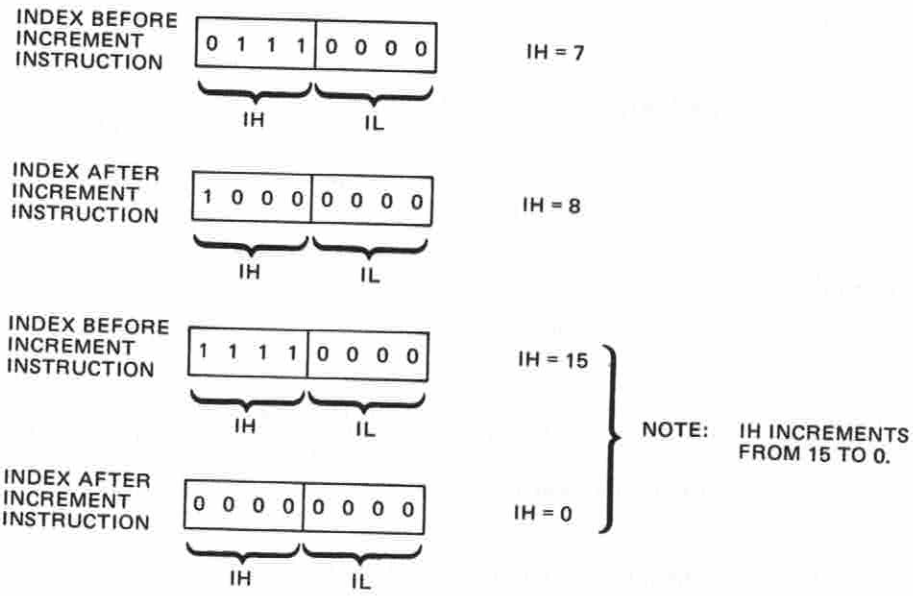
ICIL Increment IL (code 340)

Increment the contents of IL by 1. No carry into IH will occur.



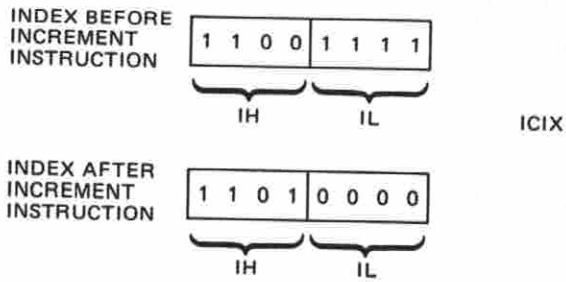
ICIH Increment IH (code 341)

Increment the contents of IH by 1



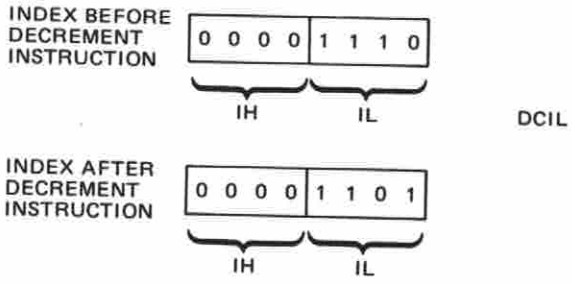
ICIX Increment IX (code 342)

Increment the contents of IX by 1. All eight bits participate. IL will carry into IH. 377 increments to 000.



DCIL Decrement IL (code 344)

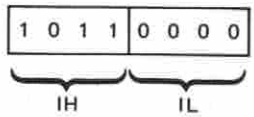
Decrement the contents of IL by 1. No borrow from IH will occur. IL = 15 decrements to IL = 0, with IH unchanged.



DCIH Decrement IH (code 345)

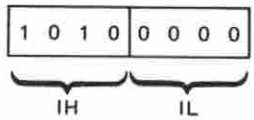
Decrement the contents of IH by 1. IH = 15 decrements to IH = 0, with IL unchanged.

INDEX BEFORE DECREMENT INSTRUCTION

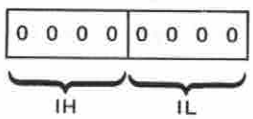


DCIH

INDEX AFTER DECREMENT INSTRUCTION

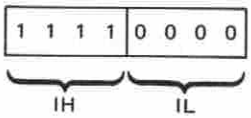


INDEX BEFORE DECREMENT INSTRUCTION



DCIH

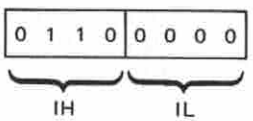
INDEX AFTER DECREMENT INSTRUCTION



DCIX Decrement IX (code 346)

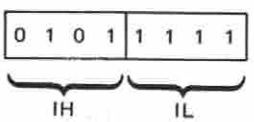
Decrement the contents of IX by 1. 000 decrements to 377.

INDEX BEFORE DECREMENT INSTRUCTION



DCIX

INDEX AFTER DECREMENT INSTRUCTION

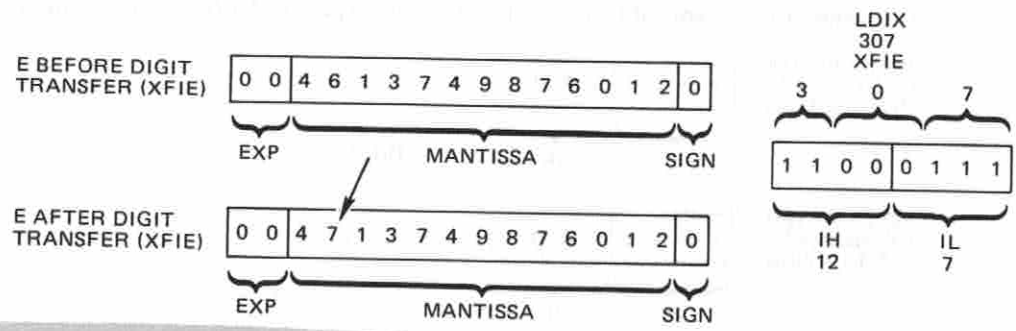


- Exchange and Transfer Instructions (Index Register with the E-Register)

XFIE Transfer Digit from IL to E (code 325)

Transfer the contents of IL to the E-register at the digit position specified by the contents of IH. Only the specified digit position of the E-register is changed.

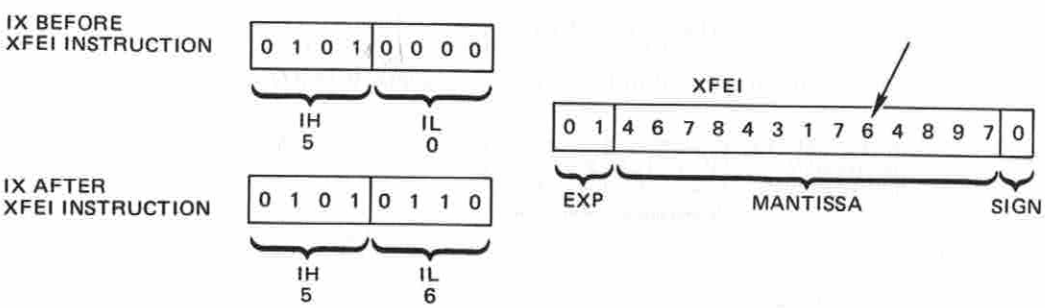
The following example loads a 7 into digit position 12.



XFEI Transfer Digit from E to IL (code 326)

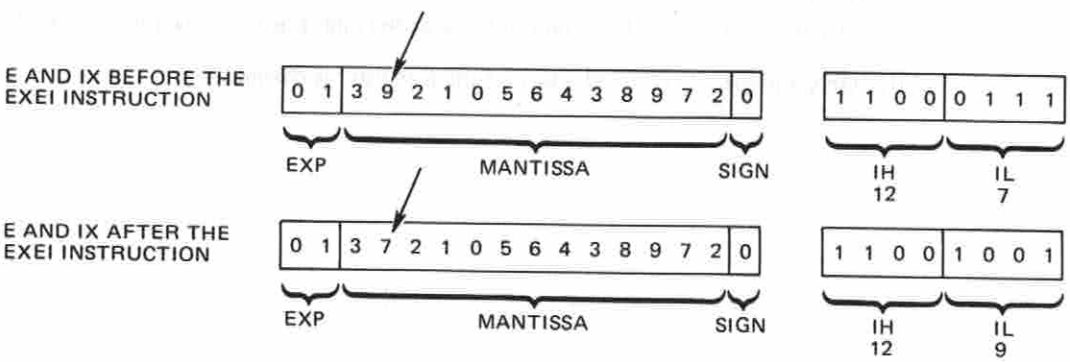
Transfer the contents of the digit position in the E-register specified by IH, to IL. Only IL is changed.

The following example transfers the contents of digit position 5 into IL. (Assume LDIX, 120 was executed to initialize IX)



EXEI Exchange IL With Digit from E (code 327)

Exchange the contents of IL with the contents of the E-register digit position specified by IH. Only IL and the specified E-register digit position are altered.



- Logical Instructions (Index Register)

Logical instructions enable the programmer to perform the Boolean operations AND, OR, and Exclusive OR.

OEIX,xxx Logical Exclusive OR (code 262)

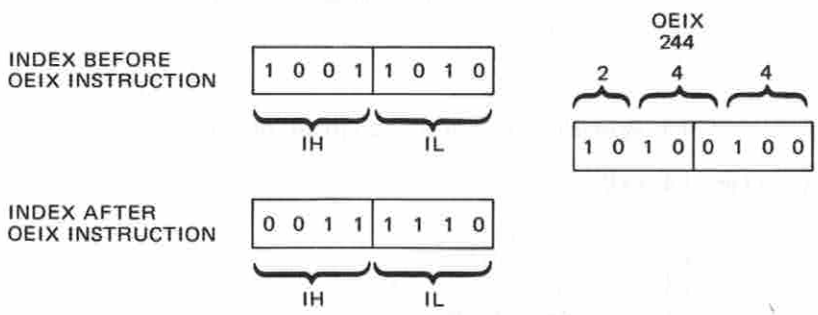
Performs a logical Exclusive OR operation between the contents of IX and the bit pattern xxx. The result is left in the index register. The Exclusive OR operation is as follows:

Exclusive OR Truth Table

Initial IX Bit	OEIX xxx Bit	Final IX Bit
0	0	0
1	0	1
0	1	1
1	1	0

(Note that a 1 in an xxx bit position will toggle the corresponding IX bit.)

The OEIX command is particularly useful for toggling flags/bits. The following example toggles H8, H2, and L4:



ORIX,xxx Logical OR (code 263)

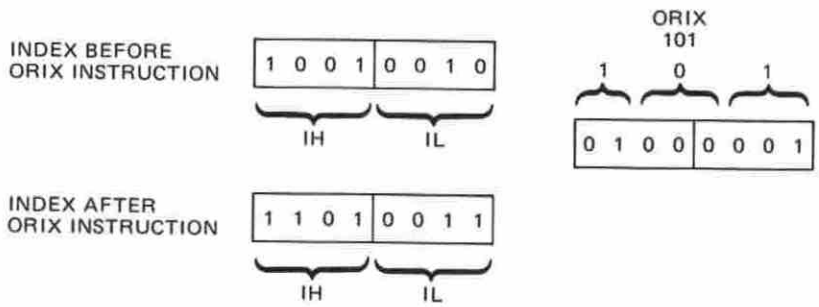
Performs a logical OR operation between the contents of the IX and the bit pattern xxx. The result is left in the index register. The OR operation is as follows:

OR Truth Table

Initial IX Bit	ORIX xxx Bit	Final IX Bit
0	0	0
1	0	1
0	1	1
1	1	1

(Note that a 1 in an xxx bit position will ensure that the corresponding IX bit becomes a 1. This is often referred to as "ORing in" a bit.)

The ORIX instruction is especially useful for setting a flag/bit. The following example ensures that bits H4 and L1 are set.



ANIX,xxx Logical AND (code 267)

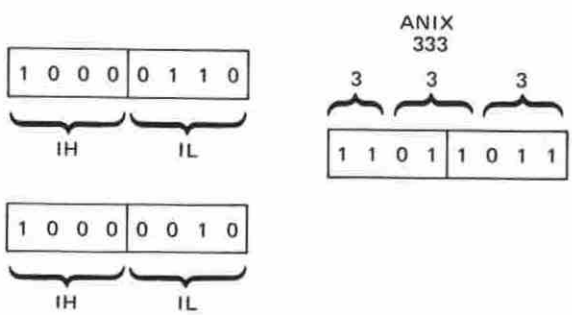
Performs a logical AND operation between the contents of IX and the bit pattern xxx. The result is left in the index register. The AND operation is as follows:

AND Truth Table

Initial IX Bit	ANIX xxx Bit	Final IX Bit
0	0	0
1	0	0
0	1	0
1	1	1

(Note that a 0 in an xxx bit position ensures that the corresponding bit in IX becomes a 0. This is often referred to as "ADDing out" a bit.)

The ANIX instruction is especially useful in clearing (resetting) flags/bits. The example below ensures that H2 and L4 are 0.

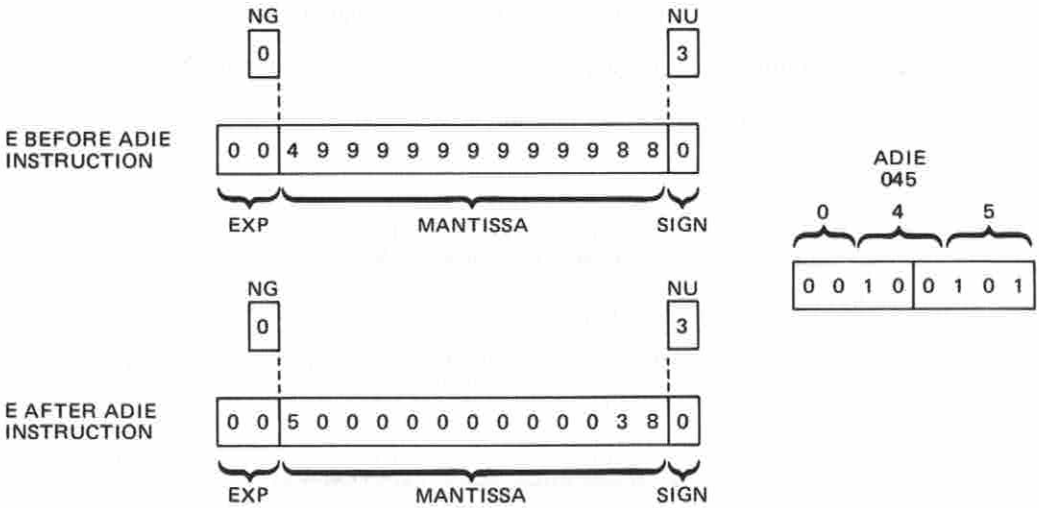


- Add Digit Instruction (Index Register and E-Register)

ADIE Add IL to E (code 324)

Add the contents of IL to the contents of the E-register digit position specified by IH. (Algebraic signs are ignored in this operation.) If IH is zero, the contents of IL are added to the contents of NU. This instruction is especially useful in rounding routines.

The following example adds a 5 to digit position 2.

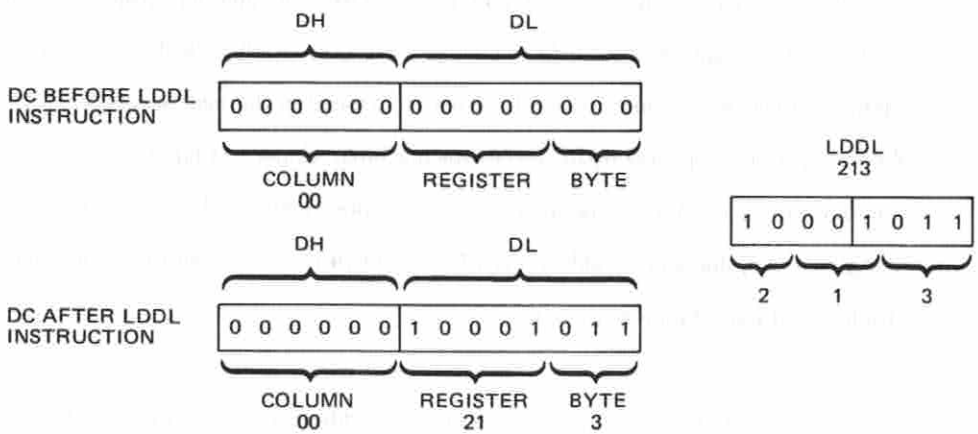


ADDRESS REGISTER MANIPULATION INSTRUCTIONS

- Initializing Instructions (Address Register)

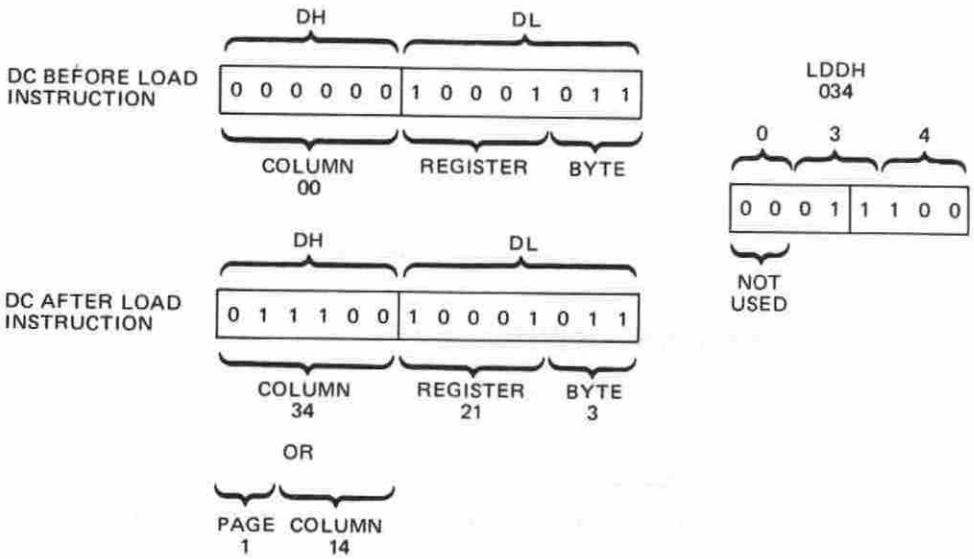
LDDL,xxx Load DL (code 271)

Load bit 7 through bit 0 of DC (figure 1-5) with the value xxx. For example, to load DL so that it specifies byte 3 in register 21, execute the double instruction LDDL, 213 as shown below:



LDDH,xxx Load DH (code 272)

Load bit 13 through bit 8 of DC with the six least significant bits of xxx. The two most significant bits of xxx are ignored, and by convention, zeroes are arbitrarily chosen. Bits 13 through 8 of DC can be considered the page/column number or the column (00 through 77) number.



Note that although this manual generally refers to DH as designating a page and a column number relative to that page (termed "page relative address", or, simply, "relative address"), it is often useful to think of this same column in terms of its location (termed "absolute address") within the full memory map, (see figure 1-2) of the machine. Under this notation, it will be seen that the four pages consist of a total of sixty-four columns, number 00 through 77 (octal). In the above example, the relative address page 1, column 14 (octal) is seen to be equivalent to the absolute address column 34 (octal).

In general it is easier to think in terms of absolute addresses when working with DC, while thinking in terms of relative addresses is more useful when working with the Branch and Jump instructions previously discussed.

LDDC,xxx Load DC (code 273)

This instruction is a shortened form of the instruction sequence LDDH, 077, LDDL, xxx.

- Store, Recall and Exchange Instructions (Address Register)

STD0 Store DC in D0 (code 170)

Store the contents of DC in register D0. The contents of DC remain unchanged.

RCD0 Recall D0 to DC (code 171)

Recall the contents of D0 to DC. The contents of D0 remain unchanged.

XCD0 Exchange DC With D0 (code 172)

Exchange the contents of DC with the contents of D0.

RCD1 Recall D1 to DC (code 174)

Recall the address from D1 to DC. The contents of D1 remain unchanged.

XCD1 Exchange DC With D1 (code 175)

Exchange the contents of DC with the contents of D1.

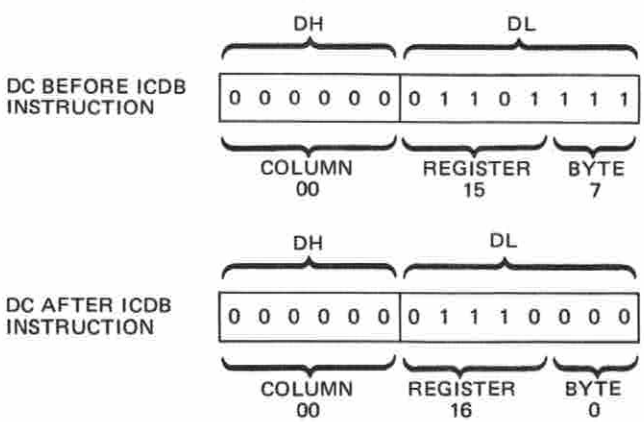
XCD1, RCD1 Store DC in D1 (Double instruction, 175, 174)

To store the contents of DC in register D1, execute XCD1 and RCD1. The contents of DC effectively remain unchanged.

- Increment and Decrement Instructions (Address Register)

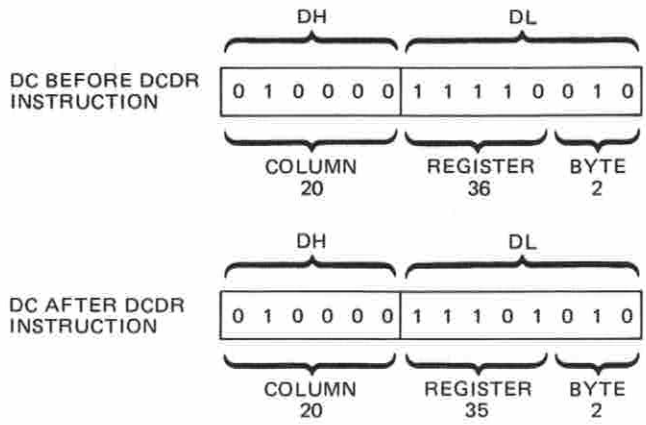
ICDB Increment DC by One Byte (code 334)

Increment the address in DC by one byte (one step). There is no carry into the page portion (two most significant bits) of DH. That is, 377 (register 37, byte 7) increments to 000 (register 00, byte 0) of the next column, except at the page boundary. For example, if DC is initially 37377, ICDB will increment it to 20000. This is referred to as "page wrap-around".



DCDR Decrement DC by One Register (code 337)

Decrement the address in DC by one register (eight bytes). Due to page wrap-around, 20000, for example, decrements to 37370.

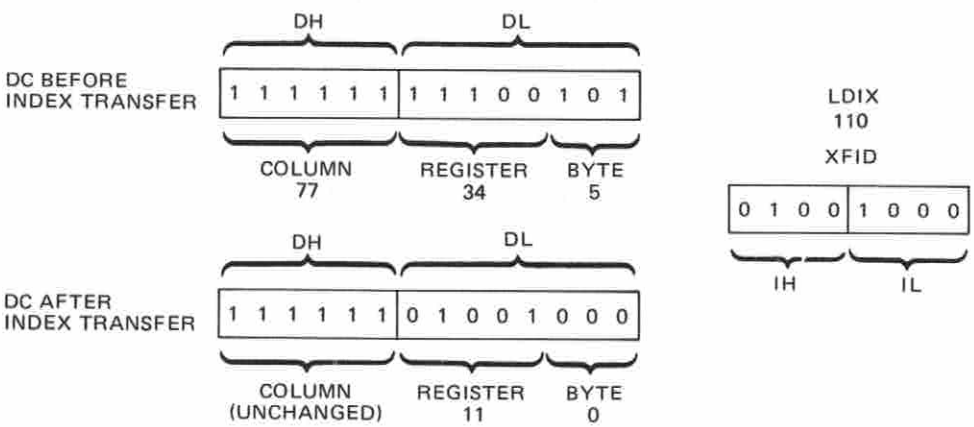


- Exchanges and Transfers between index Register and Address Register

XFID Transfer Index to DL (code 361)

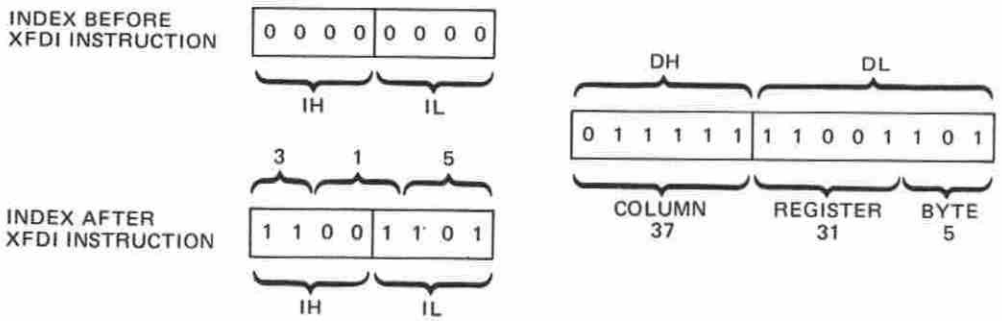
Transfers the contents of the index register to DL. The contents of the index register and DH are unchanged.

The following example loads DL to point at register 11, byte 0 of the current column.



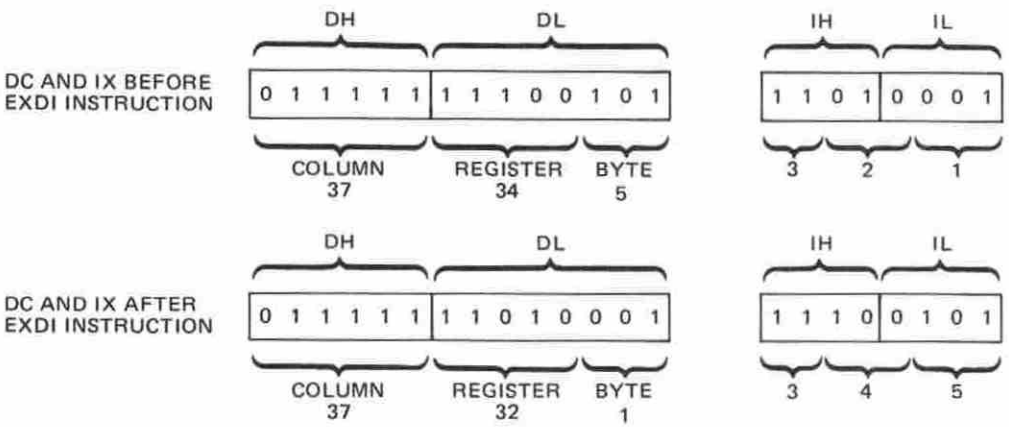
XFDI Transfer DL to Index (code 362)

Transfer the contents of DL to the index register. The contents of DH and DL are unchanged.



EXDI Exchange DL With Index (code 363)

Exchange the contents of DL with the contents of the index register. The contents of DH are unchanged.



- Miscellaneous Address Manipulation Instructions

RCLP Recall P (double instruction; code 371, 371)

Removes the top address in the P-stack (that is, the address most recently stored in the P-stack) and places it in the program counter, thereby making it the address of the next instruction to be executed. This unstacks the P-stack by one level.

STKD Stack DC (code 372)

Copy the contents of DC on the top of the P-stack. This function pushes the address already in the P-stack down one level. DC is unchanged.

RCLD Recall P to DC (code 373)

Removes the address from the top of the P-stack and places it in DC. This effectively unstacks the P-stack one level. To retrieve the top of the P-stack without unstacking that address, execute RCLD, STKD.

NOOP No Operation (code 377)

Increment the program counter and continue program execution with the next instruction in sequence.

SWITCH GROUPS, FLAG GROUPS AND THE I/O BUS

- Switch Groups

Switches are manual devices, therefore they cannot be toggled by a program.

There are two groups of eight switches in the machine, Switch Group A and Switch Group B. The instructions RSWA (Read Switch Group A) and RSWB (Read Switch Group B) read the current status of the switch group into IX. If a switch is off, the bit is 0; if a switch is on, the bit is 1.

RSWA Read Switch Group A (code 300)

Read the status of Switch Group A into IX. Switch Group A consists of one byte of information designating the position of certain internal switches. That is, if a switch is closed, the corresponding bit of IX will be set to 1 after the RSWA instruction. After the RSWA instruction has set the appropriate bits in the index register, the bits are interpreted as shown in figures 2-1a and 2-1b.

RSWB Read Switch Group B (code 301)

Read the status of Switch Group B into IX. Like Group A, Switch Group B consists of one byte of information, and a bit is set when the corresponding internal switch is closed. After the RSWB instruction has set the appropriate bits in the index register, the bits are interpreted as shown in figures 2-1a and 2-1b.

RINS Read Input Status Data (code 302)

Read the input status byte into IX. This byte contains input/outpus status information, and its interpretation may vary depending on the peripheral device used.

MODEL 1830 AND MODEL 1810

GROUP NAMES									INSTRUCTION TO READ GROUP INTO IX	INSTRUCTION TO WRITE IX INTO GROUP
SWITCH GROUP A	PRINT	LIST	SENSE II	SENSE I	AUTO POINT	ON	RUN	PAPER ADV	RSWA	
SWITCH GROUP B	Ln e^x	USED BY ROMS	USED BY ROMS	DD. MMYY	*	*	*	*	RSWB	
INPUT STATUS	BUSY	*	READY CRST	CRON	I/O REQUEST	LOAD	*	KEYR	RINS	
CARD READER	← PAPER CARD READER INPUT INSTRUCTION →								RCRD	
KEY BUFFER	KERR	← KEY CODE →							RKEY	
FLAG GROUP A	*	*	CARD LOAD LITE	IDLE LITE	USED BY ROMS	BINA	USED BY ROMS	USED BY ROMS	RFGA	WFGA
FLAG GROUP B	*	*	*	*	*	*	*	*	RFGB	WFGB

* INDICATES THAT NO FUNCTION IS PRESENTLY ASSIGNED

Figure 2-1b. Switch Groups and Flag Groups

RCRD Read Card Reader Data (code 303)

Read the card reader data byte into IX. This byte contains the bits currently transmitted by the card reader.

RKEY Read Key Data (code 304)

Read momentary key data into IX.

- Flag Groups

Flags are programmable devices; they may be set and reset by a program. There are two types of flags in the machine: hardware flags and key flags. The two types are not related.

Hardware Flags

There are two groups of 8 hardware flags: Flag Group A and Flag Group B. The instructions RFGA (Read Flag Group A) and RFGB (Read Flag Group B) put the current status of the flag group into the index. In order to change the status of a flag group, the instructions WFGA (Write Flag Group A) and WFGB (Write Flag Group B) must be executed to initiate the new flag conditions.

RFGA Read Flag Group A (code 306)

Read the status of Flag Group A into IX. The eight bits of Flag Group A are defined in figures 2-1a and 2-1b.

RFGB Read Flag Group B (code 307)

Read the status of Flag Group B into IX. Flag Group B consists of one byte; however, the bits for that group have not currently been defined.

WFGA Write Flag Group A (Double Instruction; 316, 316)

Write the contents of IX into Flag Group A.

WFGB Write Flag Group B (Double Instruction 317, 317)

Write the contents of IX into Flag Group B.

Key Flags

There are two key flags in the machine: Flag I and Flag II. These flags may be set, tested, and reset by keyboard instructions as discussed in the Programming Reference Manual for each model.

These key flags may also be set, tested, and reset using macro-instructions. Execute the following instructions to read Flag I and Flag II into IX: LDDC, 342, RCID. Flag I will now be index low bit 1 and Flag II will now be index low bit 2. Using the Index Register Data Manipulation Instructions, these flags may now be set, tested or reset. To save the new status of Flag I and Flag II, execute an STID with DC pointing to 77342.

NOTE: DO NOT MODIFY ANY OTHER BITS IN THIS INDEX.

- I/O Bus Instructions

OPCB Output Control Byte (code 314)

Output the control byte, via the output channel, from IX. The control byte will vary depending on the device used.

OPDB Output Data Byte (code 315)

Output a data byte, via the output channel, from IX.

IPDB Input Data Byte (code 305)

Input a data byte, via input channel, into IX.

III. SPECIAL TECHNIQUES

EXECUTING PROGRAM FROM MAIN DATA

In order to load program steps into main data memory, the program steps must first be loaded into program memory, Page 0. The steps are then loaded on a magnetic card. (Symbolic addresses cannot be used because they will not be defined when reloaded in Page 1.) The number of program steps to be recorded from program memory onto the magnetic card should be specified as evenly divisible by eight. This ensures that no error will occur when reading the program into main data memory. To load the program into main data (register) memory, proceed using the standard technique for entering data into registers from a magnetic card.

The following six program steps are necessary to transfer program control from Page 0 to Page 1:

```
LDDL  }
XXX   } load register/byte of first instruction
LDDH  }
XXX   } load column for first instruction
STKD
RCLP
```

Program execution continues at the step pointed to by DC. It should be noted that on Page 1, you cannot use symbolic addressing or halts. Keyboard jumps and branches will always send the program back to Page 0 addresses.

The program steps placed in main data may be a continuation of the main program. In this case, the above instruction sequence, used to access Page 1, would, with appropriate addresses, be used to return to Page 0. If the instructions in main data constitute a subroutine, the main program must branch to the six-step transfer program, above. This sets up the P-stack such that the subroutine on page 1 returns to the step following the branch to the transfer program.

STORING DATA INTO PROGRAM MEMORY

To utilize the program memory area for data storage, simply load DC to point at the location (register) of interest on Page 0 and use the STED, RCED, and XCED instructions. It is, of course, the programmer's

responsibility to see that his program does not attempt to execute the data/codes in such storage areas as if they were meaningful program instructions.

The following example stores the content of the E-register into location 00310 (equivalent to steps 200-207)

LDDH, 000, LDDL, 310, STED

PRINTING WITH A SELECTED SYMBOL

Besides the standard Print X and Print Answer functions provided on the keyboard, there is another method which can be used to print the contents of the entry register with a symbol selected by the user. The symbol to be printed is identified by the contents of IX.

Index Symbol Print (code 150) (Models 1860 and 1880)

Find the desired symbol in figure 3-1a. Add the column number from the top of the table to the row number at the right side of the table. The sum of these two numbers is the XXX code that is to be loaded into IX immediately before executing the Index Symbol Print instruction. For example, the code for an "M" is 314. To print the number in the E-register with an "M" beside it, execute LDIX, 314, 150.

Code 266, —, —.

Symbol Print Instruction (Models 1810 and 1830)

The code sequence 200, nnn, XXX may be used, in conjunction with the subroutine below, for printing selected symbols for the 1810 and 1830. Using figure 3-1b, XXX is determined as described under Index Symbol Print, for the desired symbol. The value of nnn addresses the first octal step of the following subroutine (in which the values of YYY and ZZZ load the address register with the address of the step following LDIX).

↓	RCLD	LDDH	STID	LDDL	000
	RCID	YYY	060	340	XFIE
	ICDB	LDDL	LDDH	RCED	STED
▼	STKD	ZZZ	076	LDIX	RCLP

TESTING FOR NUMERAL ENTRY

The numeral entry routine uses the byte at 77350 for controlling numeral entry. When numeral entry is initialized, IH of this byte is set to point at digit position 13. Numeral entry is considered terminated if IH of this byte is zero.

A popular technique for escaping data entry loops, without requiring the operator to signal via the Flag key or Sense switch, is to write the entry loop such that the program can automatically detect when RESUME

Figure 3-1a. Symbol Table for Models 1860 and 1880

COLUMN NUMBERS →	360	340	320	300	260
0	$\frac{1}{X}$	↓	=	0	0
1	$\sqrt{}$	↑	+	1	1
2	ℓg	↕	−	2	2
3	ℓg^{-1}	*	X	3	3
4	G	r	÷	4	4
5	∧	S	α^x	5	5
6	!	$\widehat{S}$	R°	6	6
7	SD	n-1	◊	7	7
10	Br	n	I	8	8
11	LR	A	Ju	9	9
12	X^2	(	F	)	t
13	e	Σ	$\bar{\Sigma}_0$	D	CL
14	O	X	Φ	M	X^2
15	−	Y	↑→	S	z
16	P	∓	∡	df	$\|x\|$
17					

ROW NUMBERS

Figure 3-1b. Symbol Table for Models 1810 and 1830

360	340	320	300	260
$\frac{1}{X}$	↓	=	0	0
$\sqrt{}$	↑	+	1	1
ℓg	↕	−	2	2
ℓg^{-1}	*	X	3	3
@	G	÷	4	4
∧	i	α^x	5	5
B	◊	□	6	6
SD	C	L	7	7
D	N	I	8	8
LR	A	%	9	9
X^2	(	M	)	Y
7	Σ	U	/	M
#	T	X	+	Q
E	F	V	△	D
A	P	R	T	S

Figure 3-1. Symbol Tables

is pressed without having made a keyboard entry. The following sequence of instructions test if a numeral entry was made at the Halt.

```
HALT
LDDC
350
RCID
JHNZ
XXX ──► NUMERAL ENTRY MADE (Goes to register-byte xxx)
 ▼
NO NUMERAL ENTRY MADE (Continues to next step)
```

It should be noted that this technique will not recognize that a numeral entry was made if only the digit zero was entered. This restriction can be circumvented by entering a zero value as [0] [.] i.e., zero followed by a decimal point.

FORCING TERMINATION OF NUMERAL ENTRY

Since macro-instructions will not terminate numeral entry, it is sometimes desirable to force termination of numeral entry. To do so, simply execute the following instructions:

CLIX, LDDC, 350, STID

DOLLAR SIGN PRINTOUT (Models 1810 and 1830)

A dollar sign can be printed, with one space between the dollar sign and the first digit, by branching to the following subroutine, after the appropriate number is in the E-register. The number in the E-register is lost after executing this subroutine. (If no space is desired between the dollar sign and the first digit, omit one ICIH instruction; if more than one space is desired, add an additional ICIH for each additional space.)

```
060     340     YYY
LDDH    XFEI    ICIH
076     ICIL    ICIH
LDDL    JLNZ    LDIL
350     XXX     015
RCED    DCIH    XFIE
LDIX    JU00    STED
                RCLP
```

XXX refers to the address of the first ICIH instruction.

YYY refers to the address of the XFEI instruction.

The JU00 command must be appropriately altered if this routine is not in Column 00.

APPENDIX A. INSTRUCTIONS LISTED ALPHABETICALLY

In the following table, the macro-instructions are listed alphabetically by name. (Instructions listed in the Programming Reference Manuals are not repeated here.)

Name	Mnemonic	Code	Page
Add IL to E	ADIE	324	2-18
Branch to Column 00	BR00,xxx	200	2-1
Branch to Column 01	BR01,xxx	201	2-2
Branch to Column 02	BR02,xxx	202	2-2
Branch to Column 03	BR03,xxx	203	2-2
Branch to Column 04	BR04,xxx	204	2-2
Branch to Column 05	BR05,xxx	205	2-2
Branch to Column 06	BR06,xxx	206	2-2
Branch to Column 07	BR07,xxx	207	2-2
Branch to Column 10	BR10,xxx	210	2-2
Branch to Column 11	BR11,xxx	211	2-2
Branch to Column 12	BR12,xxx	212	2-2
Branch to Column 13	BR13,xxx	213	2-2
Branch to Column 14	BR14,xxx	214	2-3
Branch to Column 15	BR15,xxx	215	2-3
Branch to Column 16	BR16,xxx	216	2-3
Branch to Column 17	BR17,xxx	217	2-3
Clear E, Including NG and NU	CLRE	330	2-7
Clear Index Register	CLIX	360	2-9
Decrement DC by One Byte	DCDB	336	2-22
Decrement DC by One Register	DCDR	337	2-23
Decrement Exponent	DCEX	353	2-9
Decrement IH	DCIH	345	2-15
Decrement IL	DCIL	344	2-14
Decrement IX	DCIX	346	2-15

Name	Mnemonic	Code	Page
Exchange DL With Index	EXDI	363	2-24
Exchange DC With D0	XCD0	172	2-21
Exchange DC With D1	XCD1	175	2-21
Exchange E According to DC	XCED	333	2-8
Exchange IH With IL	EXIX	367	2-12
Exchange IL With Digit from E	EXEI	327	2-16
Exchange IX According to DC	XCID	323	2-13
Exchange IX With I0	SCI0	162	2-11
Exchange IX With I1	XCI1	165	2-11
Increment DC by One Byte	ICDB	334	2-21
Increment DC by One Register	ICDR	335	2-22
Increment Exponent	ICEX	352	2-8
Increment IH	ICIH	341	2-14
Increment IL	ICIL	340	2-13
Increment IX	ICIX	342	2-14
Input Data Byte	IPDB	305	2-29
Jump If DC Equals D0	JDE0,xxx	252	2-6
Jump If Exponent Is Not Zero	JXNZ,xxx	256	2-6
Jump If Exponent Is Zero or Positive	JXZP,xxx	257	2-6
Jump If H1 Is One	JIH1,xxx	244	2-5
Jump If H2 Is One	JIH2,xxx	245	2-5
Jump If H4 Is One	JIH4,xxx	246	2-5
Jump If H8 Is One	JIH8,xxx	247	2-6
Jump If IH Is Not Zero	JHNZ,xxx	261	2-6
Jump If IL Is Not Zero	JLNZ,xxx	260	2-6
Jump If IX Is Not Equal to I0	JIN0,xxx	250	2-6
Jump If L1 Is One	JIL1,xxx	240	2-5
Jump If L2 Is One	JIL2,xxx	241	2-5
Jump If L4 Is One	JIL4,xxx	242	2-5
Jump If L8 Is One	JIL8,xxx	243	2-5
Jump If Mantissa Is Zero or Positive	JMZP,xxx	255	2-6

Name	Mnemonic	Code	Page
Jump If Mantissa Is Not Zero	JMNZ,xxx	254	2-6
Jump If NG is Zero	JNGZ,xxx	253	2-6
Jump to Column 00	JU00,xxx	220	2-3
Jump to Column 01	JU01,xxx	221	2-3
Jump to Column 02	JU02,xxx	222	2-3
Jump to Column 03	JU03,xxx	223	2-3
Jump to Column 04	JU04,xxx	224	2-4
Jump to Column 05	JU05,xxx	225	2-4
Jump to Column 06	JU06,xxx	226	2-4
Jump to Column 07	JU07,xxx	227	2-4
Jump to Column 10	JU10,xxx	230	2-4
Jump to Column 11	JU11,xxx	231	2-4
Jump to Column 12	JU12,xxx	232	2-4
Jump to Column 13	JU13,xxx	233	2-4
Jump to Column 14	JU14,xxx	234	2-4
Jump to Column 15	JU15,xxx	235	2-4
Jump to Column 16	JU16,xxx	236	2-4
Jump to Column 17	JU17,xxx	237	2-5
Load DC	LDDC,xxx	273	2-20
Load DH	LDDH,xxx	272	2-20
Load DL	LDDL,xxx	271	2-19
Load Index High	LDIH,xxx	265	2-9
Load Index Low	LDIL,xxx	264	2-10
Load Index Register	LDIX,xxx	266	2-10
Logical AND	ANIX,xxx	267	2-18
Logical Exclusive OR	OEIX,xxx	262	2-17
Logical OR	ORIX,xxx	263	2-17
No Operation	NOOP	377	2-25
Output Control Byte	OPCB	314	2-29
Output Data Byte	OPDB	315	2-29

Name	Mnemonic	Code	Page
Read Card Reader Data	RCRD	303	2-28
Read Flag Group A	RFGA	306	2-28
Read Flag Group B	RFGB	307	2-28
Read Input Status Data	RINS	302	2-25
Read Key Data	RKEY	304	2-28
Read Switch Group A	RSWA	300	2-25
Read Switch Group B	RSWB	301	2-25
Recall D0 to DC	RCD0	171	2-21
Recall D1 to DC	RCD1	174	2-21
Recall E According to DC	RCED	332	2-8
Recall I0 to IX	RCI0	161	2-11
Recall I1 to IX	RCI1	164	2-11
Recall IX According to DC	RCID	322	2-12
Recall P	RCLP	371, 371	2-24
Recall P to DC	RCLD	373	2-25
Shift Left in the E-Register	SHLE	351	2-8
Shift Right in the E-Register	SHRE	350	2-7
Shift Left Circular in IX	SLCI	347	2-11
Shift Right Circular in IX	SRCI	343	2-10
Stack DC	STKD	372	2-24
Store DC in D0	STD0	170	2-21
Store DC in D1	XCD1, RCD1	175, 174	2-21
Store E According to DC	STED	331	2-8
Store IX According to DC	STID	321	2-12
Store IX in I0	STI0	160	2-11
Store IX in I1	STI1	163	2-11
Transfer DL to Index	XFDI	362	2-24
Transfer Digit from E to IL	XFEI	326	2-16
Transfer Digit from IL to E	XFIE	325	2-15
Transfer Index to DL	XFID	361	2-23

Name	Mnemonic	Code	Page
Write Flag Group A	WFGA	316, 316	2-28
Write Flag Group B	WFGB	317, 317	2-28

APPENDIX B. INSTRUCTIONS LISTED NUMERICALLY

In the following table, the macro-instructions are listed numerically.

Code	Mnemonic	Name	Page
160	STI0	Store IX in I0	2-11
161	RCI0	Recall I0 to IX	2-11
162	XCI0	Exchange IX With I0	2-11
163	STI1	Store IX in I1	2-11
164	RCI1	Recall I1 to IX	2-11
165	XCI1	Exchange IX With I1	2-11
170	STD0	Store DC in D0	2-21
171	RCD0	Recall D0 to DC	2-21
172	XCD0	Exchange DC With D0	2-21
174	RCD1	Recall D1 to DC	2-21
175	XCD1	Exchange DC With D1	2-21
175, 174	XCD1, RCD1	Store DC in D1	2-21
200	BR00,xxx	Branch to Column 00	2-1
201	BR01,xxx	Branch to Column 01	2-2
202	BR02,xxx	Branch to Column 02	2-2
203	BR03,xxx	Branch to Column 03	2-2
204	BR04,xxx	Branch to Column 04	2-2
205	BR05,xxx	Branch to Column 05	2-2
206	BR06,xxx	Branch to Column 06	2-2
207	BR07,xxx	Branch to Column 07	2-2
210	BR10,xxx	Branch to Column 10	2-2
211	BR11,xxx	Branch to Column 11	2-2
212	BR12,xxx	Branch to Column 12	2-2
213	BR13,xxx	Branch to Column 13	2-2
214	BR14,xxx	Branch to Column 14	2-3
215	BR15,xxx	Branch to Column 15	2-3
216	BR16,xxx	Branch to Column 16	2-3
217	BR17,xxx	Branch to Column 17	2-3

Code	Mnemonic	Name	Page
220	JU00,xxx	Jump to Column 00	2-3
221	JU01,xxx	Jump to Column 01	2-3
222	JU02,xxx	Jump to Column 02	2-3
223	JU03,xxx	Jump to Column 03	2-3
224	JU04,xxx	Jump to Column 04	2-4
225	JU05,xxx	Jump to Column 05	2-4
226	JU06,xxx	Jump to Column 06	2-4
227	JU07,xxx	Jump to Column 07	2-4
230	JU10,xxx	Jump to Column 10	2-4
231	JU11,xxx	Jump to Column 11	2-4
232	JU12,xxx	Jump to Column 12	2-4
233	JU13,xxx	Jump to Column 13	2-4
234	JU14,xxx	Jump to Column 14	2-4
235	JU15,xxx	Jump to Column 15	2-4
236	JU16,xxx	Jump to Column 16	2-4
237	JU17,xxx	Jump to Column 17	2-5
240	JIL1,xxx	Jump If L1 Is One	2-5
241	JIL2,xxx	Jump If L2 Is One	2-5
242	JIL4,xxx	Jump If L4 Is One	2-5
243	JIL8,xxx	Jump If L8 Is One	2-5
244	JIH1,xxx	Jump If H1 Is One	2-5
245	JIH2,xxx	Jump If H2 Is One	2-5
246	JIH4,xxx	Jump If H4 Is One	2-5
247	JIH8,xxx	Jump If H8 Is One	2-6
250	JIN0,xxx	Jump if IX Not Equal to I0	2-6
252	JDE0,xxx	Jump If DC Equals D0	2-6
253	JNGZ,xxx	Jump If NG is Zero	2-6
254	JMNZ,xxx	Jump If Mantissa Is Not Zero	2-6
255	JMZP,xxx	Jump If Mantissa Is Zero or Positive	2-6
256	JXNZ,xxx	Jump If Exponent Is Not Zero	2-6
257	JXZP,xxx	Jump If Exponent Is Zero or Positive	2-6
260	JLNZ,xxx	Jump If IL Is Not Zero	2-6

Code	Mnemonic	Name	Page
261	JHNZ,xxx	Jump If IH Is Not Zero	2-6
262	OEIX,xxx	Logical Exclusive OR	2-17
263	ORIX,xxx	Logical OR	2-17
264	LDIL,xxx	Load Index Low	2-10
265	LDIH,xxx	Load Index High	2-9
266	LDIX,xxx	Load Index Register	2-10
267	ANIX,xxx	Logical AND	2-18
271	LDDL,xxx	Load DL	2-19
272	LDDH,xxx	Load DH	2-20
273	LDDC,xxx	Load DC	2-20
300	RSWA	Read Switch Group A	2-25
301	RSWB	Read Switch Group B	2-25
302	RINS	Read Input Status Data	2-25
303	RCRD	Read Card Reader Data	2-28
304	RKEY	Read Key Data	2-28
305	IPDB	Input Data Byte	2-29
306	RFGA	Read Flag Group A	2-28
307	RFGB	Read Flag Group B	2-28
314	OPCB	Output Control Byte	2-29
315	OPDB	Output Data Byte	2-29
316, 316	WFGA	Write Flag Group A	2-28
317, 317	WFGB	Write Flag Group B	2-28
321	STID	Store IX According to DC	2-12
322	RCID	Recall IX According to DC	2-12
323	XCID	Exchange IX According to DC	2-13
324	ADIE	Add IL to E	2-18
325	XFIE	Transfer Digit from IL to E	2-15
326	XFEI	Transfer Digit from E to IL	2-16
327	EXEI	Exchange IL With Digit from E	2-16
330	CLRE	Clear E, Including NG and NU	2-7
331	STED	Store E According to DC	2-8
332	RCED	Recall E According to DC	2-8

Code	Mnemonic	Name	Page
333	XCED	Exchange E According to DC	2-8
334	ICDB	Increment DC by One Byte	2-21
335	ICDR	Increment DC by One Register	2-22
336	DCDB	Decrement DC by One Byte	2-22
337	DCDR	Decrement DC by One Register	2-23
340	ICIL	Increment IL	2-13
341	ICIH	Increment IH	2-14
342	ICIX	Increment IX	2-14
343	SRCI	Shift Right Circular in IX	2-10
344	DCIL	Decrement IL	2-14
345	DCIH	Decrement IH	2-15
346	DCIX	Decrement IX	2-15
347	SLCI	Shift Left Circular in IX	2-11
350	SHRE	Shift Right in the E-Register	2-7
351	SHLE	Shift Left in the E-Register	2-8
352	ICEX	Increment Exponent	2-8
353	DCEX	Decrement Exponent	2-9
360	CLIX	Clear Index Register	2-9
361	XFID	Transfer Index to DL	2-23
362	XFDI	Transfer DL to Index	2-24
363	EXDI	Exchange DL With Index	2-24
367	EXIX	Exchange IH With IL	2-12
371, 371	RCLP	Recall P	2-24
372	STKD	Stack DC	2-24
373	RCLD	Recall P to DC	2-25
377	NOOP	No Operation	2-25