William Barden, Jr.

TRS-80

Assembly Language Subroutines

A collection of easy-to-use subroutines for your TRS-80

$18.95 A SPECTRUM BOOK

Harold May
Dec '86
2^{00}

WILLIAM BARDEN, JR.

TRS-80 ASSEMBLY LANGUAGE SUBROUTINES

A SPECTRUM BOOK

PRENTICE-HALL, INC., Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging in Publication Data

Barden, William T.
TRS-80 assembly language subroutines.

(A Spectrum Book)
1. TRS-80 (Computer)—Programming. 2. Assembler language (Computer program language) I. Title.
QA76.8.T18B373 001.64'2 82-383
ISBN 0-13-931188-2 (pbk.) AACR2

This Spectrum Book is available to businesses and organizations at a special discount when ordered in large quantities. For information, contact Prentice-Hall, Inc., General Publishing Division, Special Sales, Englewood Cliffs, N. J. 07632.

0-13-931188-2

A SPECTRUM BOOK

10 9 8 7 6 5 4 3 2

Printed in the United States of America

Editorial production supervision by Frank Moorman
Cover design by Ira Shapiro
Manufacturing buyers: Cathie Lenard and Barbara A. Frick

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA PTY. LIMITED, *Sydney*
PRENTICE-HALL CANADA, INC., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL OF SOUTHEAST ASIA PTE. LTD., *Singapore*
WHITEHALL BOOKS LIMITED, *Wellington, New Zealand*

Contents

III APPENDICES

Preface

Radio Shack TRS-80 Model I, II, and III assembly language is a powerful way to program. Assembly-language programs may run as much as 300 times faster than their BASIC counterparts, turning a boring BASIC game into a high-speed video chase or a day-long sort into minutes. Unfortunately, assembly language is also difficult to learn and, once learned, a tedious language in which to program.

What is the solution in using assembly language on the Radio Shack computers? This book offers one solution—precanned, debugged, and documented assembly-language subroutines for the TRS-80 computers. In it, you'll find subroutines that will speed up your graphics by a factor of 300, subroutines that enable you to perform high-speed sorts, general-purpose subroutines that will allow you to do number base conversions and square roots, and special utility subroutines, such as subroutines to "dump" the video screen to cassette or to read a disk sector.

There are 65 of these assembly-language subroutines. The subroutines may be easily interfaced to BASIC programs—they are specifically geared to BASIC interfacing, as a matter of fact. Each subroutine is *relocatable*; the assembly-language code is such that the subroutine may be placed anywhere in memory without reassembling the subroutine. To make this task very easy, we've included the equivalent decimal code after the listing of each subroutine. It's simply a matter of taking the dozen, or two dozen, or three dozen decimal values and embedding them in BASIC programs as DATA statement values or strings. From that point on, the subroutine exists as part of the BASIC program.

Of course, you may not want to always use the subroutines in BASIC programs. You may want to CALL them in your own assembly-language code. We've also made it easy for you to do this. Each set of code can be called as a separate assembly-language module. You may want to reassemble and modify the code, but, if not, the code is usable as it stands, and it is completely relocatable.

Although the subroutines are slanted toward the TRS-80 Model I and III, many of them can also be used on the TRS-80 Model II; all three computers, of course, use the Z-80 microprocessor.

The first chapter of this book, "A Brief Look at TRS-80 Assembly-Language Programming," contains introductory material on Z-80 assembly-language programming, to make you familiar with some of the techniques. It's not absolutely necessary that you read this chapter. The next chapter, "Using Assembly Language on the TRS-80," shows you how assembly language may be used in either a BASIC or stand-alone environment. This chapter is not an absolute requirement, either, but you may want to study it further when you start using the subroutines and embedding them in BASIC programs or running them as separate entities.

The bulk of the book consists of 65 separate assembly-language subroutines. Each subroutine consists of a description, the subroutine listing, and equivalent decimal values for the "machine code" of the subroutine.

The description gives a brief idea of what the subroutine accomplishes and shows the input and output *parameters* that are used to pass information back and forth between the subroutine and the calling program.

The description also includes a complete explanation of the *algorithm* used in the subroutine—how the subroutine accomplishes the function in Z-80 code.

Another element in the description is a sample call to the subroutine using actual input and output values. The sample calls use a "TRS-80 Assembly-Language Subroutines Exerciser" program, TALSEX for short. TALSEX is a Model I/III Disk BASIC program that was used to exercise the subroutines; it is fully described in Chapter 2 and is used in the descriptions to conveniently show the action of each subroutine.

Notes pertaining to the use of the subroutine are also included in the description along with a "checksum" value that can be used to verify that you have entered the program data correctly.

The assembly-language listing is the actual listing from the Z-80 assembler. It shows every instruction used in the subroutine and also is heavily "commented." Because of this, the listing may be used in self-study on assembly-language programming and techniques.

The last portion of each subroutine is a complete set of decimal values to be used for inclusion in a BASIC program in DATA statements or the like. We've done the conversion from hexadecimal to BASIC for you, to minimize operator error. These values, when added together by the CHKSUM subroutine, should correspond to the Checksum value in the description, giving you a way to check the validity of the data in your program.

An appendix on Z-80 instructions and a second on decimal/hexadecimal conversion complete the book.

We hope that you'll find these subroutines useful in BASIC, in assembly-language programs, and in self-study of Z-80 assembly language on the TRS-80s.

To John Foster and "ASHEE"

I

TRS-80 ASSEMBLY-LANGUAGE PROGRAMMING TECHNIQUES

1

A Brief Look at TRS-80 Assembly-Language Programming

In this chapter we'll discuss some rudimentary assembly-language concepts. It isn't necessary that you understand everything in this chapter, or even that you read the chapter to use the subroutines in this book. If you choose to do so, however, you'll get a better idea of how assembly language is done.

The Z-80 Microprocessor

The Z-80 microprocessor is used in the TRS-80 Model I, II, and III microcomputers. It is a third-generation microprocessor that is truly a "computer on a chip." When we speak about TRS-80 assembly-language programming we're really discussing the built-in *instruction set* of the Z-80 microprocessor.

Unlike BASIC statement execution, the Z-80 performs instructions at the most rudimentary level. Typical instructions would add two 8-bit numbers, subtract two 8-bit numbers, load a CPU register with the contents of a memory location, or store a CPU register into a memory location.

All assembly-language programs are built up of a set of Z-80 instructions in sequence, which are executed by the Z-80. These instructions are held in memory in binary and may be one to four bytes long. The binary values for the instructions are called *machine language*, because this is the form that the Z-80 computing machine recognizes.

Z-80 Registers

Before we look at some of the Z-80 instructions, let's take a further look at the Z-80 *architecture*. Figure 1-1 shows the internal *registers* available to the machine-language or assembly-language programmer. We won't show some of the other registers involved in internal microprocessor operations, such as memory access or timing.

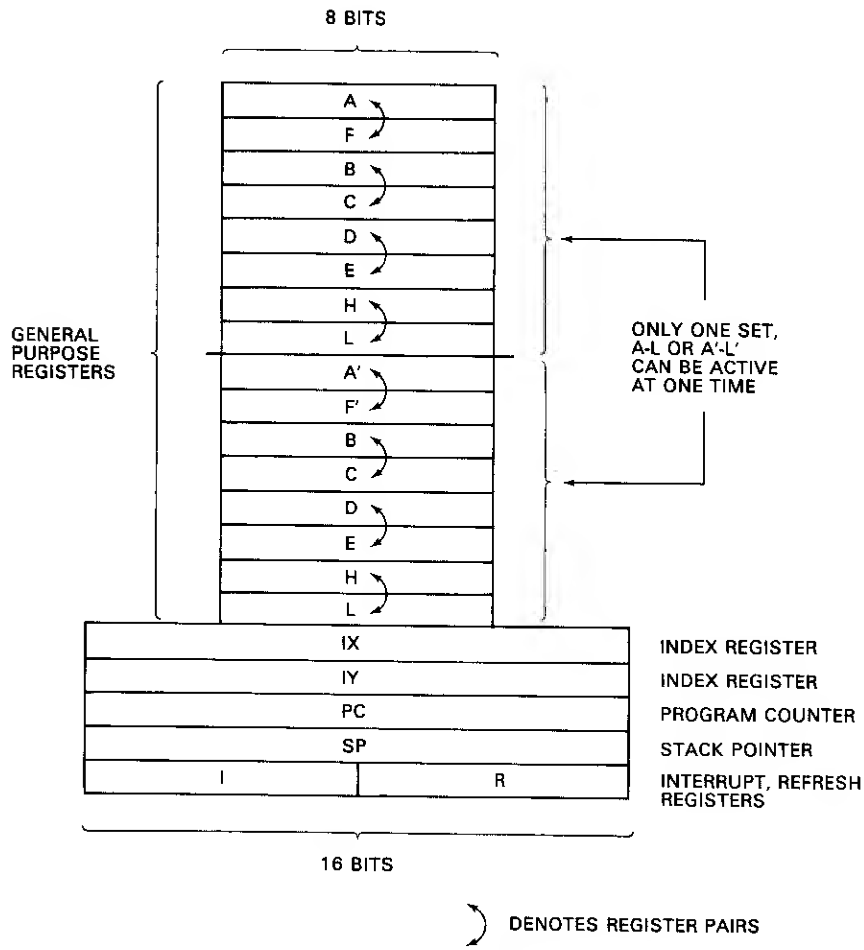


FIGURE 1-1 Z-80 registers for use in assembly language.

The Z-80 registers are fast-access memory locations located in the Z-80. The A, B, C, D, E, H, and L registers are *general-purpose* 8-bit registers in the Z-80. They are used to hold temporary results and for processing.

The A register is the main *accumulator* register. It holds one operand for adds, subtracts, and other arithmetic operations while the other operand may come from memory or another register. The other registers are used as auxiliary registers, with the exception of H and L.

H and L, along with B and C and D and E, can be grouped together as *register pairs* of 16 bits. When this is done, the registers act as three 16-bit wide registers called HL, BC, and DE. The HL register pair (often called the HL register) is a kind of 16-bit accumulator similar to the A register. It can be used for 16-bit adds, subtracts, and other operations.

The IX and IY registers are 16-bit registers that can be used as *index registers*, or pointers to memory locations. We'll discuss these a little later on, when we talk about Z-80 addressing modes.

The PC, or program counter, register is the main control register not only in the Z-80 microprocessor, but in the whole TRS-80 system. It controls execution of all programs, assembly-language or BASIC. After all, BASIC is simply an assembly-language program that operates on a series of higher-level statements. The PC is 16 bits wide and points to the first byte of the next instruction in memory to be executed. As an assembly-language program executes, the PC is constantly being updated by one to point to the next byte of the instruction or is loaded with a *jump address* to enable a jump to a new location in memory.

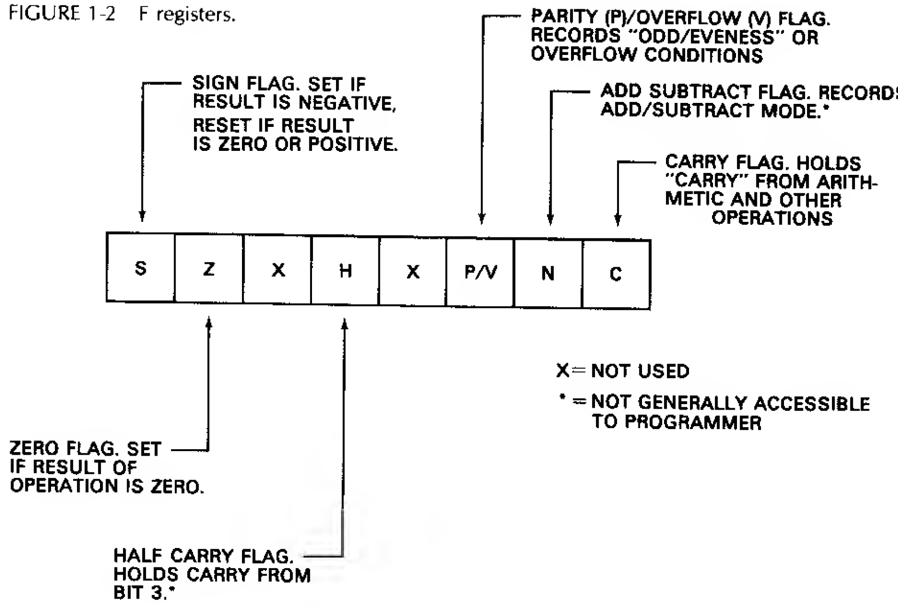
The SP, or stack pointer, register, is a 16-bit register that points to the *stack area*. The stack area is a special section of RAM memory that is set aside to hold return addresses from CALL instructions, temporary results, or interrupt locations. This stack area, typically only one hundred bytes long, builds downward as the stack is used. Every time an assembly-language CALL instruction (similar to a BASIC GOSUB) is executed, the return address from the PC register is *pushed* onto the stack. A subsequent RET(urn) instruction *pops* the stack and reloads the PC with the return address.

The R and I registers can be largely ignored by the programmer. (The R register is used in one subroutine in this book.) The I register is used for a special interrupt mode in other Z-80 systems, and R is used for *refresh* of the dynamic memories in the TRS-80 systems.

We've given a thumbnail sketch of all of the Z-80 registers except one, the F register. The F register is a collection of the eight *flags* shown in figure 1-2. These flags are set by the action of assembly-language instructions. The Z flag, for example, stands for Z(ero) flag. The Zero flag is set whenever the result of certain adds, subtracts, or other types of arithmetic operations is zero. The other flags are set for similar conditions. The flags are used in *conditional jump* instructions to alter the flow of an assembly-language program. The program could jump to a new set of codes if the result of an add was a negative number, for example. The A and F registers are treated together as one 16-bit register pair for storage in the stack and other operations.

The seven general-purpose registers and the flags register are duplicated in the Z-80. The second set, called the prime set, is available as additional register storage. One or the other set may be selected by two instructions.

FIGURE 1-2 F registers.



Z-80 Instructions

The *instruction repertoire* of the Z-80 contains well over 700 unique instructions. Fortunately, many of these instructions can be grouped together, and the actual number of similar groups is much easier to manage.

Loads generally load the contents of an 8-bit memory location, CPU register, or *immediate value* in the instruction itself into a CPU register. A second class of loads *store* the contents of an 8-bit CPU register into memory. Loads may also be done on 16-bits of data in a register pair, loading or storing two bytes of data. There are a great number of load-type instructions in the Z-80. A load instruction in the Z-80 is denoted by an "LD," and you will see many, many loads in every program. A load is really just a way of transferring data.

Arithmetic instructions add or subtract 8 bits of data with the A register, or 16 bits of data with the HL, IX, or IY registers. These are simply adds and subtracts of binary numbers, sometimes with the state of the *Carry* flag (a one or a zero) being added into the result. Adds and subtracts are denoted by ADD, ADC, SUB, or SBC. A special type of subtract, the compare (CP), compares two 8-bit values.

A number of instructions related to arithmetic instructions allow adding (INCrementing) or subtracting (DECrementing) one count from the contents of a CPU register or memory location.

Logical instructions perform ANDs, ORs, or exclusive ORs on operands in the A register. The ANDs and ORs are identical to BASIC ANDs and ORs, except that they operate with 8 bits of data, while the XOR is similar to an OR except that two one bits produce a zero bit in the result.

Shift instructions shift data in any of the 8-bit CPU registers one bit position right or left. There are several different types of shifts, including the rotate, which rotates the data out of the register and into the other end, the logical shift, which shifts data out with zeroes filling vacated bit positions, and the arithmetic shift, which *sign extends* the value in the register. Mnemonics for shifts are RLCA, RLA, RRCA, RRA, RLC, RL, RRC, RR, SLA, SRA, SRL, RLD, and RRD.

Jumps, CALLs, and return instructions handle alterations of the program path similar to BASIC GOTOs, IF . . . THEN, GOSUBs, and RETURNs. There are two types of jumps, conditional and unconditional. Unconditional jumps *always* jump to a new location, while a conditional jump jumps *if* the condition, such as Zero Flag=1, is present. CALLs are identical to BASIC GOSUBs. They call an assembly-language subroutine and save the return point in the program stack. A RET(urn) retrieves the return address from the stack and returns to the instruction after the CALL. CALLs and RETurns may also be conditional or unconditional. Jumps are denoted by JP or JR, CALLs by CALL, and RETurns by RET.

A special type of jump is used in conjunction with a loop count in the B register. The DJNZ instruction (Decrement and Jump if Not Zero) decrements the count in B by one and then jumps back to the beginning of a loop if the count is not zero.

Bit manipulation instructions allow operations on a bit level. Data in a CPU register or in memory can be referenced by the bit address, 7 through 0, and the applicable bit can be set, reset, or tested. Bit manipulation instructions are denoted by SET, RES, or BIT.

"Block" instructions allow operations on many bytes of data in a block. Blocks of data may be searched (CPI, CPD, CPIR, CPDR) or moved (LDI, LDD, LDIR, LDDR) using these instructions.

Input/output instructions handle operations between CPU registers and an external input/output device, such as cassette tape. The TRS-80s allow both "memory-mapped" and "I/O mapped" input/output. This means that an input/output device may look either like another memory location (memory mapped) or as a special device addressed through an input/output *port*. When the system I/O ports are used, input is normally done with an IN instruction and output with an OUT instruction.

Stack instructions allow data in CPU register pairs; including the AF register pair, to be temporarily stored in the system stack. PUSH pushes a single register pair to the stack and POP retrieves the data into the original register pair or another.

We haven't mentioned all of the Z-80 instructions, but the above list would encompass most of the instructions used in common Z-80 assembly-language code. Special instructions are sometimes described in the documentation on the subroutines, and there's always reference material in Zilog or Radio Shack publications that describe the Z-80 instructions in great detail.

Z-80 Addressing Modes

There are a number of different ways to access data with the Z-80 instruction set. These are called *addressing modes*.

One type of addressing mode allows operations between CPU registers. You can see that it's convenient to add two numbers located in two CPU registers, for example. A complete instruction using this type of addressing mode might be "ADD A,B," which adds the contents of the B register to the contents of the A(ccumulator) register and puts the result into the A register. Another sample of this type of instruction is "INC DE," which adds one to the contents of the DE register pair and puts the result back into the DE register pair.

Register addressing is normally used for arithmetic and logical instructions, shifts, and load instructions.

Load and store instructions must transfer data between CPU registers and memory. One addressing mode that implements this in load-type instructions is the *direct addressing* mode. This mode allows a CPU register to be loaded or stored directly to a RAM memory address specified in the instruction. A "LD A,(3C00H)," for example, would load the contents of the first video display memory location into the A register. Similarly, a "LD (3FFFH),A" would store the contents of A into the last location of the video display memory. Not only 8 bits of data can be transferred. Sixteen-bit operations are possible with instructions such as "LD (3C00H), HL," which stores the contents of the HL register pair into video memory locations 3C00H (L) and 3C01H (H).

Direct addressing is also used in some types of jump and CALL instructions. In this case the address specified in the instruction is the address to which the instruction will jump or which the instruction will call. The instruction "CALL 212H," for example, CALLs the ROM subroutine located at memory location 212H. The 212H is a part of the instruction as a direct address.

The *immediate addressing* mode is used to load a data value into either an 8-bit CPU register or into a 16-bit register pair. The data value is usually a constant value when loaded into the 8-bit register, but is often an address value when loaded into a 16-bit register pair. The term "immediate" means that the data is present as part of the instruction itself. The advantage to this mode is that of speed and convenience. The immediate mode is faster than accessing a data value from a memory location and one does not have to keep track of a large number of constants in memory. The following code loads the value of 41H (ASCII "A") into the A register, and the address 3C00H into the HL register pair:

```
LD      A,41H          ;load "A" into A
LD      HL,3C00H       ;load start video memory to HL
```

Notice that when immediate addressing is used, the data is not surrounded by parentheses, as it is in direct addressing, where the data represents a memory address. The exception to this is in the jump or CALL instructions where the memory address for the jump or CALL does not have parentheses.

Another type of *memory reference* addressing mode uses a register pair as a pointer to a location in memory. The most commonly used pointer is the HL register pair. In this type of addressing, the HL, BC, or DE register is preloaded (by another instruction) with the address of the memory location to be used in the "register indirect" instruction. An example of this would be the two instructions

```
LD      HL,3C00H        ;load video memory start
LD      (HL),A          ;store into video start
```

The first instruction loads the memory address of 3C00H (the first byte of the video memory) into the HL register pair. The next instruction stores the contents of the A register by a "register indirect" store, using the memory address in the HL register pair.

Another type of addressing mode that is similar in concept to that of using the register pairs as pointers is the *indexed addressing* mode. In this mode, the IX or IY index register is used as a pointer to a memory location. The index register by itself, however, does not represent the complete address of the memory location. The *effective address,* the one used in the instruction, is formed by adding the contents of the IY or IX index register together with a *displacement address* in the indexed instruction. The displacement is a "signed" binary value of 8 bits that may be a positive or negative quantity. The effective address, therefore, is larger or smaller than the address in the index register. The indexed addressing mode is commonly used where the index register points to the beginning or end of a table or list of data; the displacement in the instruction can then be used to reference memory locations close to the address in the index register.

Suppose, for example, we had a table of data at memory location 8000H. The following code would load 8000H + 5 into the A register, and 8000H + 10 into the B register:

```
LD      IY,8000H        ; load index register with 8000H
LD      A,(IY+5)        ; load 8005H contents into A
LD      B,(IY+10)       ; load 800AH contents into B
```

One important addressing mode for our purposes is the *relative addressing* mode. In this mode, the memory address is not present in the instruction, as it was for the jump or CALL, but is *relative* to the location of the instruction itself. A displacement value in the instruction is used by the CPU, along with the contents of the program counter, to figure out the effective address for the jump. For example, if we looked in the machine-language code for a "DJNZ" instruction, we would not see a two-byte memory address, but a one-byte displacement value. If the jump in the DJNZ was to be made back to location 8000H, and the DJNZ was at location 800AH, the displacement value would be 0F4H, a negative 0CH or twelve (the program counter points to two more than the start of the DJNZ instruction).

Relative addressing is important for our purposes because it makes *relocatable code* possible—assembly-language code that can be moved around anywhere in memory and still execute properly. The key to relocatability is to avoid direct addresses within instructions, and relative jumps such as DJNZ and JRs are used to advantage.

Bit addressing is another type of addressing mode. This mode is used only for the bit-processing instructions. The bit position within a byte is referenced in this mode, along with one of the other addressing modes we've mentioned above. To set bit 6 in the memory location pointed to by the HL register pair, for example, we'd have

```
BIT     6,(HL)     ; set bit 6 in memory location
```

Bit positions in 8-bit bytes are numbered from left to right, bit 7 through bit 0. Bit positions in 16-bit "words" are numbered from left to right also, bit 15 through bit 0. The bit position number represents the power of two associated with the bit.

There are no hard and fast rules about which addressing type to use. Many times the choice is dictated by the instruction—not all addressing types are permitted with every instruction.

Machine Code and Assembly Language

We talked briefly about machine code, but haven't really made a distinction between machine and assembly code. The difference can be seen quite easily by reference to a typical listing in this book.

Figure 1-3 shows a short listing for CHKSUM. The listing is divided into several parts. Starting from the left, we have the memory locations, in hexadecimal, for which the subroutine was assembled. The value for each line shows where the instruction on the line will reside: The code always starts at location 7F00H. In the case of subroutines in this book, these locations are meaningless, as the code can be used not only at locations 7F00H, but 8000H, 888FH, 9013H, or any place in memory the user cares to put them. (More on that in Chapter 2.)

The next column is the actual machine code for the instruction in hexadecimal. Two hexadecimal digits (0 through 9, A through F) make up one byte, so you can see that the machine code is from two to six hexadecimal characters or one to three bytes long. The maximum length of an instruction is four bytes, or eight hexadecimal digits. Note that the memory location for the instruction in the first column reflects the size of the previous instruction. If an instruction is three bytes long and is located at 7F0BH, for example, the next memory location will be three bytes greater, or 7F0EH.

The third column shows the editing line number for the instruction. The editing line numbers are used only during the editing process and are never used during program loading or execution.

The fourth, fifth, sixth, and seventh columns represent the assembly-language code for the instructions. Sometimes this portion is called the "source image," because this is the portion that appears in the *source file* that is assembled.

The fifth column is the mnemonic for the instruction *operation code,* or *op-code*. We've been using mnemonics all along. They are just a shorthand way of writing down the instruction in convenient and recognizable form. The operation code describes the primary function of the instruction, as, for example, an "ADD."

FIGURE 1-3 Partial CHKSUM listing. SOURCE IMAGE

```
7F00                00100          ORG      7F00H          ;0522
                    00110 ;***********************************************************
                    00120 ;* CHECKSUM MEMORY. CHECKSUMS A BLOCK OF MEMORY.        *
                    00130 ;*     INPUT: HL=>PARAMETER BLOCK                         *
                    00140 ;*            PARAM+0,+1=STARTING ADDRESS OF BLOCK        *
                    00150 ;*            PARAM+2,+3=# OF BYTES IN BLOCK              *
                    00160 ;*     OUTPUT:HL=ADDITIVE CHECKSUM                        *
                    00170 ;***********************************************************
                    00180 ;
7F00  F5            00190 CHKSUM   PUSH   AF          ;SAVE REGISTERS
7F01  C5            00200          PUSH   BC
7F02  D5            00210          PUSH   DE
7F03  DDE5          00220          PUSH   IX
7F05  CD7F0A        00230          CALL   0A7FH       ;***GET PB LOC'N***
7F08  E5            00240          PUSH   HL          ;TRANSFER HL TO IX
7F09  DDE1          00250          POP    IX
7F0B  DD6E02        00260          LD     L,(IX+2)    ;GET # OF BYTES
7F0E  DD6603        00270          LD     H,(IX+3)
7F11  DD5E00        00280          LD     E,(IX+0)    ;GET STARTING ADDRESS
7F14  DD5601        00290          LD     D,(IX+1)
7F17  D5            00300          PUSH   DE          ;TRANSFER TO IX
7F18  DDE1          00310          POP    IX
7F1A  010100        00320          LD     BC,1        ;DECREMENT VALUE
7F1D  AF            00330          XOR    A           ;CLEAR CHECKSUM
7F1E  DD8600        00340 CHK010   ADD    A,(IX+0)      ;CHECKSUM
7F21  DD23          00350          INC    IX            ;BUMP ADDRESS PNTR
7F23  B7            00360          OR     A             ;CLEAR CARRY
7F24  ED42          00370          SBC    HL,BC         ;DECREMENT COUNT
7F26  20F6          00380          JR     NZ,CHK010     ;GO IF NOT DONE
7F28  6F            00390          LD     L,A         ;MOVE CHECKSUM TO HL
7F29  2600          00400          LD     H,0
7F2B  DDE1          00410          POP    IX          ;RESTORE REGISTERS
7F2D  D1            00420          POP    DE
7F2E  C1            00430          POP    BC
7F2F  F1            00440          POP    AF
7F30  C39A0A        00450          JP     0A9AH       ;***RETURN STATUS***
7F33  C9            00460          RET                ;NON-BASIC RETURN
0000                00470          END
00000 TOTAL ERRORS
```

MEMORY LOCATIONS — MACHINE CODE — EDITING LINE # — INSTRUCTION LABEL — OP CODE — OPERANDS — COMMENTS

The sixth column is the *operands* column. The column is used to show which operands will take part in the instruction. The instruction at CHK010, for example, ADDs the location pointed to by the IX index register plus a displacement of 0 to the contents of the A register. The formats for the operands are relatively fixed and can be found in other reference materials for Z-80 assembly language.

The fourth column is the *label* of the instruction. This is an optional column, but really delineates the difference between machine language and *symbolic* assembly language. The label is used by the assembler program in lieu of a memory address. The instruction at 7F26H in figure 1-3, for example, refers not to a jump address at 7F1EH, but to a *label* of "CHK010." The assembler translated the label reference to the proper address in the instruction, in this case, a relative displacement.

The last column on the listing is the *comments* column. This column contains descriptive text about the use of the instruction. Note that we've indented the comments column to show *loops*. Each level of loops is indented two spaces, and there may be as many as three levels of loops. Also in the comments column, we've marked certain instructions with asterisks. These represent instructions which may be ignored under "stand-alone" conditions when the subroutine is not used with BASIC. This is explained fully in Chapter 2.

Additional Z-80 Assembly-Language Materials

As the title of this chapter indicated, we've briefly discussed Z-80 assembly language. If you would like a more in-depth discussion of instruction formats, addressing modes, and assembly-language techniques, we suggest you obtain the reference manual for the Zilog Z-80 microprocessor, or refer to the instruction manual for the Radio Shack Editor/Assembler, which reproduces much of the same material. The author's Radio Shack book, "TRS-80 Assembly-Language Programming," is also a good place to start.

In the next chapter we'll discuss some of the general techniques of using assembly language, and specific details about the use of the subroutines in this book.

2
Using Assembly Language on the TRS-80s

In this chapter we'll look at some of the techniques involved in using assembly language on the TRS-80 Models I, II, and III, especially in regard to interfacing the machine-language representation of assembly-language code with BASIC programs.

Using the Model I and III Assemblers

There are a number of editor/assemblers for the Model I and III computers, and they are very similar. All are modifications of the basic Radio Shack cassette-based Editor/Assembler. The following description of the assembly process will use the Radio Shack Editor/Assembler as a point of reference; material on disk files will refer to the various modifications available for the Radio Shack Editor/Assembler to enable it to read and write source and object files on disk.

This material is offered in case you wish to assemble some of the subroutines in the book and modify them for your own use; let's stress once again that you can use the subroutines in the book without ever touching an assembler.

Editing the Source File

The first step in assembly is to edit the source file. Let's use another short subroutine as an example. The SQROOT subroutine is shown in figure 2-1. To start the edit, the assembler is loaded from cassette or disk. The SYSTEM command is used to load from cassette. Loading from disk simply involves entering "EDTASM" followed by ENTER.

```
        ORG     7F00H               ;0522
;*********************************************************
;* SQUARE ROOT. CALCULATES INTEGER PORTION OF SQUARE     *
;* ROOT OF A GIVEN NUMBER.                                *
;*     INPUT: HL=NUMBER                                   *
;*     OUTPUT:HL=INTEGER PORTION OF SQUARE RT OF NUMBER   *
;*********************************************************
;
SQROOT  PUSH    BC                  ;SAVE REGISTERS
        PUSH    DE
        CALL    0A7FH               ;***GET NUMBER***
        LD      B,0FFH              ;INITIALIZE RESULT
        LD      DE,-1               ;FIRST ODD SUBTRAHEND
SQR010  INC     B                     ;INCREMENT RESULT COUNT
        ADD     HL,DE                 ;SUBTRACT ODD NUMBER
        DEC     DE                    ;FIND NEXT ODD NUMBER
        DEC     DE
        JR      C,SQR010              ;CONTINUE IF NOT MINUS
        LD      L,B                 ;GET RESULT
        LD      H,0                 ;NOW IN HL
        POP     DE                  ;RESTORE REGISTERS
        POP     BC
        JP      0A9AH               ;***RETURN ARGUMENT***
        RET                         ;NON-BASIC RETURN
        END
```

FIGURE 2-1 Sample Source file for edit.

The "I" command is used to enter a new file. The "I" command is the insert command, and is normally used to insert lines between existing lines in an edit file. In this case, however, there are no existing lines and the "I" command starts a new set of lines with the starting number 100 and line increment of 10.

The "source image" text of the subroutine can now be entered. Each line is typed in its entirety and an ENTER is used to terminate a line. The first several lines look like this:

```
*I
00100           ORG     7F00H       ;0522
00110   ;*********************************************************
00120   ;* SQUARE ROOT. CALCULATES
```

The left arrow key can be used to backspace to correct errors in entry. Other editing features are very similar to the BASIC line editor—such things as "L" for line, "S" for search, and so forth. After the entire text has been entered, the BREAK is pressed. This terminates the insert mode and displays the greater than prompt.

The source text is now in memory. The source text can be written out to cassette by the command "W SQROOT." This command produces a *source file* with the name SQROOT. A subsequent "L SQROOT" enables the source file to be read in from cassette as a text file.

The source text can be written out to disk as a source file by the command "WD SQROOT/SRC" ("W D=SQROOT/SRC" in some versions). If this is done, the text will be transferred to disk as a source file and can be read in for further editing at any time by a "LD SQROOT/SRC" (LD=SQROOT/SRC).

After the source file has been created on disk or cassette, it can be reloaded as a check on its validity, or you can simply work with the text in memory.

Assembling the Source File

To assemble the SQROOT subroutine, type "A/NO/WE/NS" followed by ENTER. The source file will now assemble and the listing will be displayed on the screen. If there are any errors in the text, the Editor/Assembler will stop and any key may be pressed to restart the assembly. At the end of the listing you'll see a message that looks like this:

```
00000 TOTAL ERRORS,
```

indicating that there were no assembly errors. The "/X" entries were "switch options" calling for "No Object," "Wait on Error," and "No Symbol Table Listing."

What has been produced up to this point? The machine code was generated, but it was simply part of the listing that was rapidly displayed on the screen. All we've done to this point was to assemble and display the listing on the screen to check for errors. If everything is all right, we can proceed. Otherwise, the errors in the source file can be corrected, another assembly done, and the process repeated until we get a "clean" assembly. Many errors will relate to instruction format, and these can be corrected by reference to the Radio Shack Editor/Assembler manual. There are also slight quirks in some of the assembler versions—such things as "(IY+0)" not assembling and "(IY)" assembling properly. We can't detail all of these here. It's a shame they exist; try to work around them!

When we have a clean assembly, we can create an *object file* and save it on disk. The object file is really a machine-language version of the program, with a "header" for the disk file and other data pertinent to the load. Most of the content on the disk file will be the actual machine-language code that you see on the listing. To create the object file, assemble without the "No Object" switch, which is the default mode of the assembly. You may also assemble to line printer, while you're at it:

```
*A/LP/NS
```

The Editor/Assembler version may ask for a "destination" (disk or tape) and for a file name before the assembly. As we've used SQROOT/SRC for the source file, we might use SQROOT/OBJ for object. The assembly will proceed as before, except that the object file will be written to cassette or disk.

Loading the Object File

At this point we have both the source file and object file on cassette or disk. The source file is saved for possible modification. The object file can now be loaded and executed. To load the object file from cassette, the SYSTEM mode is used once again to load the file named at assembly time.

To load the object file from disk, we must first get back to the Disk Operating System, and then use the LOAD command:

```
*B
DOS READY
LOAD SQROOT/OBJ
DOS READY
```

The object file is located by the LOAD command but it is not executed. It is just as well, as we were not set up properly to execute the SQROOT program. Where is SQROOT loaded? The ORG command establishes the starting point for the program, which in all cases in this book is 7F00H. The ORG command can be modified to make the load point compatible with your system; just put in a new argument in place of 7F00H. If you want a square root subroutine at 0F000H in a 48K Model I, for example, reassemble with "ORG 0F000H." It may also be necessary to protect the memory area in which the object program was loaded by responding with one less than the ORG point when BASIC asks the question "MEMORY SIZE?".

Now that we have the program loaded, what do we do with it? We'll answer that question in the last part of the chapter in which we'll show you an easier way to work with the subroutines in this book when they are interfaced to BASIC.

Using the Model II Assembler

The edit, assembly, and load process is similar for the Model II. The Model II, however, uses the Radio Shack Disk Assembler, which is a more sophisticated editor/assembler. There is also a version of the Radio Shack Disk Assembler available for the Model I and III. Use of this assembler is beyond the scope of this book. The author's Radio Shack book "More TRS-80 Assembly-Language Programming," goes into some detail on the Disk Assembler.

Keying In the Object Code Directly

The assembly process can be bypassed completely by working with the object code alone and T-BUG (Radio Shack's Debug package for cassette-based systems) or DEBUG (Radio Shack's Disk Debug Package). A DEBUG utility is also present on the Model II system. The result can be saved on cassette or as a disk "core image" file. Let's see how this can be done by using the DEBUG program on a disk-based system.

The modify memory command "M" in DEBUG can be used to enter the data one byte at a time. The format of the M command is "MHHHH space," where HHHH is the hexadecimal address for the start of the memory area. Choose any memory area that is nonconflicting with TRSDOS or BASIC and in which you'd like the subroutine to reside. Now go to the listing and key in each byte in hexadecimal, following each byte with a space, and the last byte with an ENTER. The process is shown in figure 2-2, where a portion of SQROOT has been keyed into the memory area starting at 9000H.

FIGURE 2-2 Keying in object code using DEBUG.

```
AF  = 5B 08 ----1---
BC  = 0A 53 =>  B7 CA 55 09 21 5E 09 E5  CD 55 09 1B 1A 4F C8 21
DE  = 01 04 =>  1A 4D 45 4D 4F 52 59 20  53 49 5A 45 00 52 41 44
HL  = 00 54 =>  01 01 5B 1B 0A 1A 08 18  09 19 20 20 0B 78 B1 20
AF' = FF FF SZ1H1PNC
BC' = 51 5B =>  C4 CF 51 10 DE C1 C9 ED  5B 60 40 13 E5 AF ED 52
DE' = 02 02 =>-
HL' = 51 00 =>  C6 02 FF CB 02 F7 10 32  E7 20 32 01 C7 43 04 F7
IX  = 40 15 =>  01 9C 43 20 30 00 4B 49  07 58 04 31 3E 20 44 4F
IY  = 00 00 =>  F3 AF C3 74 06 C3 00 40  C3 00 40 E1 E9 C3 9F 06
SP  = 41 CA =>  52 04 C3 4B DD 03 15 40  FF FF 18 43 3F 3F 4C 00
PC  = 00 60 =>  0B 78 B1 20 FB C9 31 00  06 3A EC 37 3C FE 02 D2
        9000 => |C5 D5 CD 7F 0A 06| 20 72  65 70 65 61 74 65 64 20
9006-   9010 =>  75 6E 74 69 6C 20 77 65  20 67 65 74 20 61 20 22
20-FF   9020 =>  63 6C 65 61 6E 22 20 61  73 73 65 6D 62 6C 79 2E
        9030 =>  20 4D 61 6E 79 20 65 72  72 6F 72 73 20 77 69 6C-

NEXT BYTE FOR 9006H          SIX BYTES KEYED IN
                             AT 9000H-9005H
```

The machine code values shown on the listings do not have to be modified unless the subroutine will not be used in conjunction with BASIC. In this case, substitute the 00H code (a "NOP" instruction) for each *byte* of the starred instructions. The hexadecimal machine code is relocatable and can be used anywhere in memory.

After the data has been keyed in, perform a "G66" to reboot TRSDOS and dump the memory area by a "DUMP" command as follows:

DUMP (START=X'SSSS',END=X'EEEE')

where SSSS is the starting address in hexadecimal and EEEE is the ending address in hexadecimal.

The memory image will now be written out as a "core image module" with the file extension "/CIM." It can be loaded by the TRSDOS LOAD command in the same fashion as the assembly object file.

Using Assembly Language with Model I and III BASIC

There are two general approaches to using assembly-language code with BASIC. The first of these uses two modules, an object code module and a BASIC program module loaded at separate times. The second method embeds the machine-language code in BASIC statements which then become part of the BASIC program.

The "Two-Module" Approach

Let's look at the "two module" approach first. In this approach, the object program from assembly or debug dump is loaded first with TRSDOS. Then the BASIC interpreter is loaded and the memory area in which the object program was loaded is protected with the "MEMORY SIZE?" response. Now the BASIC program can call the assembly-language subroutine at will.

How the BASIC program calls the machine code is slightly different between Level II BASIC and Disk BASIC. Level II requires that the address of the machine code be put into locations 16526 and 16527. All addresses in the Z-80 are stored, least significant byte followed by most significant byte; so a typical sequence to establish the call address for Level II BASIC might be as follows for a machine-language program at 7F00H:

```
100 POKE 16526,0        'least significant byte
110 POKE 16527,127      'most significant byte
```

In Disk BASIC on the Model I or III, the call address is established in simpler fashion. The address of the machine-language subroutine is assigned a number from 0 to 9. A DEFUSR statement is then used to establish the address:

```
100 DEFUSR0=&H7F00
```

where &H is the prefix for hexadecimal.

Once the address is established, the machine-language subroutine can be called by a BASIC USR statement of the form A=USR(M) for Level II or A=USRn(M) for Disk BASIC. The n in the Disk BASIC version stands for the id number from 0 through 9. The M is an integer argument that can be automatically passed to the machine-language subroutine. The A is an integer argument that is passed back from the machine-language subroutine. Either or both of these arguments can be "dummies" if no arguments need to be passed.

To see how the complete sequence works, let's call the SQROOT subroutine. Assume that it has been loaded at 7F00H and BASIC has protected memory by a "MEMORY SIZE? 32511." We see from the listing that the SQROOT subroutine takes a 16-bit number and computes the integer square root, passing the argument back in HL. The following code would set up the call address in Level II BASIC, make the call, and return the result for printing:

```
100 POKE 16526,0        'least significant byte
110 POKE 16527,127      'most significant byte
120 INPUT X%            'input square
130 Y=USR(X%)           'call machine lang SQROOT
140 PRINT X%,Y          'print square, root
```

The sequence for Disk BASIC would be similar:

```
100 DEFUSR0=&H7F00      'address
110 INPUT W%            'input square
```

```
120 Z= USR0(W%)           'call machine lang SQROOT
130 PRINT W%,Z            'print square, root
```

In both cases, the argument passed to the SQROOT subroutine was the integer variable in the USR call. The argument passed back was the variable equated to the USR call.

In some subroutines, no arguments are required, or only one argument is needed. In these cases either a dummy argument, such as 0, may be used, or a variable that is not used elsewhere may be used. The SCDOWN subroutine, for example, scrolls the screen down one line and requires no input or output arguments. The call (assuming that the address has been set up) would be:

```
200 A= USR0(0)      'scroll screen down
```

and the A variable would be ignored.

Embedding Machine Language in BASIC

The second method for interfacing BASIC and assembly language is to embed the machine-language code in BASIC. There are a number of methods for doing this.

Taking the example of the SQROOT subroutine, let's look at one method that uses DATA values. The decimal values for the machine-language code of SQROOT is placed into a DATA statement:

```
100 DATA 197,213,205,127,10,6,255,17,255,255,4,25,27
110 DATA 27,56,250,104,38,0,209,193,195,154,10,201
```

The DATA values are then moved to a known area of memory on the first pass through the BASIC code. Let's use 7F00H again:

```
120 FOR I=0 TO 24        'loop
130 READ A               'read DATA value
140 POKE 15212+I,A       'store value
150 NEXT I               'loop 25 times
```

After the loop is done, the DATA values have been moved to the 7F00H area, and the machine-language code can be called in the usual fashion after setting up the address in 16526,16527 or with a DEFUSRn statement. This procedure will work with all of the subroutines in this book.

Is there a way to avoid using a predefined area, a way to make the procedure more automatic? Yes, with qualifications. Machine-language code can be embedded in strings, arrays, and even BASIC statements, but there may be some problems with this method. Again taking the SQROOT subroutine as an example, let's construct a string of machine-language values and then call the string. We can set up the string by:

```
100 A$=CHR$(197)+CHR$(213)+CHR$(205). . . . +CHR$(201)
```

One statement can be used if the number of characters in the line does not exceed the maximum line length of 255 characters. If there is not enough room in one line, two strings can be established and the two can then be concatenated into a third.

Where is the machine-language code in this case? It's somewhere in the string variable region at the top of memory. We can find out where it is by using the VARPTR function. The VARPTR function will return the location of the *string parameter block*. The string parameter block holds the length of the string and the string address as shown in figure 2-3. We can then put the string address into locations 16526, 16527 or use it in a DEFUSRn statement. A sample call of SQROOT using this technique is shown here:

```
100 A$=CHR$(197)+CHR$(213)+CHR$(205)+ . . . +CHR$(201)
110 B=VARPTR(A$)        'get string parameter block location
120 POKE 16526,PEEK(B+1)
130 POKE 16527,PEEK(B+2)
140 A=USR(M)
```

where M is the square and A is the square root returned.

For Disk BASIC, the sequence would be similar:

```
100 A$=CHR$(197)+CHR$(213)+CHR$(205)+ . . . +CHR$(201)
110 B=VARPTR(A$)
120 C=PEEK(B+1)+PEEK(B+2)*256
130 IF C>32767 THEN C=C-65536
140 DEFUSR0=C
150 A=USR0(M)
```

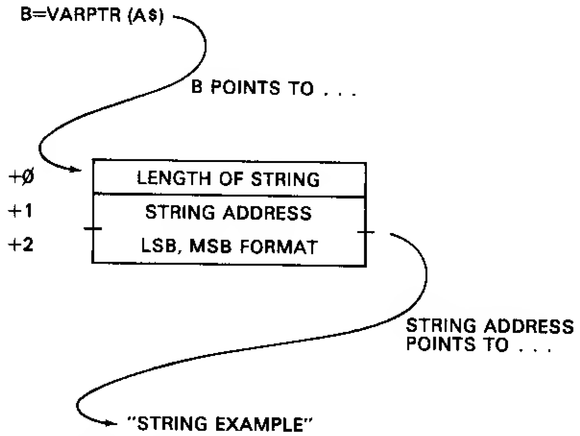


FIGURE 2-3 String parameter block format.

The IF . . . THEN statement is necessary because of a quirk of BASIC. It does not handle addresses well as integer arguments, and the subterfuge above is necessary to "fool" the interpreter into thinking that the 16-bit memory address is a signed integer value.

Now, there's one strong bit of advice that we must give. If you use the above method, be aware that everything in BASIC moves! Any time that BASIC encounters a new variable, a new array, or computes a new string, variables are readjusted. Periodically, string variables are "cleaned up," and this is done at unpredictable times. Therefore, when using the VARPTR to find the address of a string, do so only directly before the USR call, and make certain that no new variables are introduced in the call.

There are other methods similar to the above for embedding machine language in BASIC code. They all rely on using VARPTR to find the location of a string or array. The string could be a dummy string in a program statement, for example. The string

```
100 A$="THIS IS A DUMMY STRING!!!"
```

has 25 characters and can accommodate the 25 bytes of the SQROOT subroutine. Another advantage of this approach is that in this case the string is at a fixed location in memory—as long as the program statements do not change (no edits allowed). The machine-language values can be picked up from DATA statements and stored in the dummy string, and a VARPTR could then be used to find the dummy string location.

Another method is to establish a large array by a statement similar to DIM AA(100). DATA values can now be stored in the array and a VARPTR done with the first element of the array to find the start of the contiguous area for the array. (Don't try this on string arrays!)

```
100 B=VARPTR(AA(0))
```

Here again, do not introduce any new variables after finding the VARPTR address or the address will be incorrect. (New variables are placed before the array areas and the array areas are moved down!)

In the subroutines that follow we will assume that they are located in 7F00H. If you wish to use one of the methods described above to embed the machine-language code in your programs, that is perfectly feasible as long as you follow the rules. However, be careful of variables that move and things that go bump in the RAM!

Passing Multiple Arguments

In many of the subroutines in this book, it's necessary to pass more than one argument to the subroutine and back from the subroutine. Take the MOVEBL, or Move Block, subroutine. MOVEBL moves a block of memory from one area of memory to another area of memory. Three parameters are involved—the address of the existing block (the "source" address), the address of the "destination," and the number of bytes to move. All are 16-bit values.

The USR calling sequence allows only one 16-bit value to be passed. How do we pass three 16-bit addresses? The way we have established as a standard for the subroutines in this book is to pass the address of a "parameter block." The parameter block holds the necessary parameters in a predefined order. The parameter block may be anywhere in memory, either at a fixed location or in a string or array. As an example, assume that the MOVEBL subroutine is located at FF06H. The parameter block could be six bytes before, starting at 0F000H, and we'd have this Disk BASIC calling sequence:

```
100 DEFUSR0=&HF006              'address of subroutine
110 POKE 61440-65536,0          'source address=8000H
120 POKE 61441-65536,128
130 POKE 61442-65536,0          'destination address=9000H
140 POKE 61443-65536,144
150 POKE 61444-65536,0          '256 bytes
160 POKE 61445-65536,1
170 A=USR0(61440-65536)         'move block
```

In this BASIC code, we first defined the address of the subroutine as 0F006H by the DEFUSR0. Next we POKEed the source address into 0F000H and 0F001H, least significant byte followed by most significant byte (0,128 becomes 128*256+0=8000H). Then we POKEed the destination address into 0F002H and 0F003H (0,144 becomes 144*256+0=9000H). Next, we POKEed the number of bytes into 0F004H and 0F005H (0,1 becomes 1*256+0=256). Finally, we called the subroutine by the USR0 call with the input argument equal to the start of the parameter block at 61440 (0F000H). Note that we had to use the trick of subtracting 65,536 from the addresses in order to use the POKE and USR statement with BASIC integer values.

Alternatively, you could put the arguments in a dummy CHR$ string or dummy string and use VARPTR to find the string address, or you could put the arguments in an array and use VARPTR to find the first element of the array. (Just follow the rules, and make certain that no new variables are introduced after the VARPTR finds the address!)

Using Assembly Language on the Model II

The general approach for the Model II is virtually identical to that used on the Models I and III. The calling sequence uses the DEFUSRn and USRn formats of Model I/III Disk BASIC. The major difference is in the Model II's approach to passing arguments to the machine-language subroutine and back to the BASIC program.

Two system subroutines, FRCINT and MAKINT, are used in place of the machine-language code in place of ROM subroutines at 0A7FH and 0A9AH. If you are using these subroutines on a Model II together with a BASIC program, you may reassemble with the calling sequence given in the Model II BASIC reference manual. The two calling sequences would be substituted in place of the "starred" "CALL 0A7FH" or "JP 0A9AH." If you are not using a BASIC program, then many of the subroutines in this book may be used "stand alone" by replacing the starred instruction bytes with zeroes (NOPs).

How to Use the Subroutines in This Book

Now we come to the most important part of these two chapters—how do we use the subroutines in this book?

To use any of the 65 subroutines, follow this procedure:

1. Read the description of the subroutine. See if it can be used on your system. Note what parameters are involved and how large (8 or 16 bits) each one is.
2. If the subroutine is to be used without BASIC and called from your own assembly-language code (including Model II code), reassemble the subroutine to create your own source file, or create a machine-language core image module using T-BUG or BASIC. *Put a 00H byte in every instruction byte that is marked with asterisks.* This NOPs the calls to BASIC ROM routines that pass parameters. (On reassemblies, leave out these instructions.)
3. If the subroutine is to be embedded in BASIC, put the decimal values into DATA statements, and write the BASIC code to move the subroutine to a fixed area or variable area as outlined above.
4. Call the subroutine from BASIC or your own assembly-language code with the proper number of arguments. The subroutine may require no arguments, in which case dummy arguments would be used in BASIC. The subroutine may require one input argument, in which case the USRn call would specify a single integer argument. The subroutine may require one output argument, in which case the USRn call would specify a dummy input argument with a valid output argument. The subroutine may require multiple arguments, in which case the USRn call would specify the address of the parameter block containing the arguments. In assembly-language calls, the arguments are also held in a parameter block pointed to by the HL register pair.

Here are some additional rules:

1. For assembly-language calls only: HL contains the single argument on input, the single output argument, or the address of the parameter block.
2. For assembly-language calls only: Most subroutines save all registers. The ones that do not are clearly denoted.
3. For assembly-language calls only: The stack pointer is assumed initialized before the call.
4. All subroutines have relocatable code.
5. All listings have been assembled at 7F00H. The ORG point must be changed if you are reassembling at a specific area for a "two module" load. If you are using only the machine code, it is correct as it stands.
6. Certain assemblers have minor bugs in instruction formats; instructions may not assemble properly. The assembler used in these subroutines corrects some of the assembly errors. If your assembler does not assemble the source code as listed, your assembler may be flawed!
7. Error checking in these subroutines is minimal. In other words, it may be easy to blow up the system with improper arguments. This was done to keep the subroutines short. Checks should be made for proper arguments before calling the subroutine.

8. Every effort was made to keep the subroutines relocatable. Some of the resulting code may not be good programming practice in nonrelocatable code. So be it.

9. We have purposely stayed away from ROM subroutine calls because of the possibility of ROM changes. Those ROM calls that are used are clearly marked.

10. Tables have generally been avoided because of relocatability problems resulting in linear code. Here again, this may not be code to emulate in non-relocatable environments.

11. Nested subroutines within the subroutines have been avoided because of relocatability problems resulting in linear code. Again, this was done for relocatability.

12. Names of subroutines and labels are nonconflicting. You may assemble all subroutines together en masse without fear of duplicate labels on assembly.

13. All loops are indented in the comments column. Each level of loop is indented two spaces. Block moves and compares are essentially loops and are indented.

TALSEX: TRS-80 Assembly-Language Subroutines Exerciser Program

Figure 2-4 shows the complete listing of TALSEX. It is a Model I/III Disk BASIC program that we have used to exercise (and hopefully exorcise) all of the subroutines in this book. You will probably not want to use TALSEX, but we'll describe how it works in case some of the code is helpful in your BASIC interfacing. All of the sample calls for the subroutines are the output of one test case of TALSEX.

TALSEX first asks for the name of the subroutine. The name is then displayed on the screen and printed on the system printer. Next, TALSEX asks for the value to be put into HL. If no argument is required, ENTER may be pressed, otherwise the argument value is entered.

Next, the parameter block location is entered. This may be any area in free memory. If multiple arguments are being used in the subroutine, the HL value corresponds to the parameter block location. The values to be put into the parameter block are then input in the form N,V. (N is 0, 1, or 2.) If N is 1, the following value V will be 8 bits long. If N is 2, the following value V will be 16 bits long. An input of 0,0 terminates the input.

Next, TALSEX asks for a memory block location. If the subroutine uses a memory block, this value is input, otherwise ENTER is pressed. Values are then entered into the memory block as required. The memory block may be anywhere in free memory. A 0,0 input terminates the operation. A second memory block location may then be input, and values stored in this block.

Now, TALSEX asks for a location at which the assembly-language subroutine should be located. TALSEX assumes that the subroutine is currently in memory at 7F00H (from a LOAD operation in DOS). When this value is input, TALSEX moves the subroutine from the 7F00H area to the specified memory area to test relocatability.

The subroutine is then called with HL containing the specified value, and the parameter block and two memory blocks containing the specified data.

On return, the input and output values for HL, the parameter block, and the memory blocks are displayed and printed.

FIGURE 2-4 TALSEX listing.

```
1000 CLS: PRINT "TRS-80 ASSEMBLY LANGUAGE SUBROUTINES EXERCISER"
1005 DIM IO(49)
1010 PRINT:PRINT:LPRINT:LPRINT
1015 HL=70000: PB=70000: M1=70000: M2=70000: ZI=0
1017 FOR I=0 TO 49: IO(I)=-1: NEXT I
1020 A$="NAME OF SUBROUTINE": PRINT A$;: LPRINT A $;"? ";
1030 INPUT A$: LPRINT A$
1040 A$="HL VALUE": PRINT A$;: LPRINT A$;"? ";
1050 A$="": INPUT A$: LPRINT A$
1055 IF A$="" GOTO 1070
1060 HL=VAL(A$): IF HL>32767 THEN HL=HL-65536
1070 A$="PARAMETER BLOCK LOCATION": PRINT A$;: LPRINT A$;"? ";
1080 A$="": INPUT A$: LPRINT A$
1085 IF A$="" GOTO 1220
1090 PB=VAL(A$): IF PB>32767 THEN PB=PB-65536
1100 A$="PARAMETER BLOCK VALUES?": PRINT A$: LPRINT A$
1200 ZA=HL: GOSUB 10000
1220 A$="MEMORY BLOCK 1 LOCATION": PRINT A$;: LPRINT A$;"? ";
1230 A$="": INPUT A$: LPRINT A$
1235 IF A$="" GOTO 1320
1240 M1=VAL(A$): IF M1>32767 THEN M1=M1-65536
1250 A$="MEMORY BLOCK 1 VALUES?": PRINT A$: LPRINT A$
1260 ZA=M1: GOSUB 10000
1270 A$="MEMORY BLOCK 2 LOCATION": PRINT A$;: LPRINT A$;"? ";
1280 A$="": INPUT A$: LPRINT A$
1285 IF A$="" GOTO 1320
1290 M2=VAL(A$): IF M2>32767 THEN M2=M2-65536
1300 A$="MEMORY BLOCK 2 VALUES?": PRINT A$: LPRINT A$
1310 ZA=M2: GOSUB 10000
1320 A$="MOVE SUBROUTINE TO": PRINT A$: LPRINT A$;"? ";
1330 INPUT A$: LPRINT A$
1340 SL=VAL(A$): IF SL>32767 THEN SL=SL-65536
1350 FOR I=32512 TO 32767
1360 POKE(SL+I-32512),PEEK(I)
1370 NEXT I
1380 DEFUSR0=SL
1390 H1=USR0(HL)
1395 IF SL<0 THEN SL=SL+65536
1400 A$="SUBROUTINE EXECUTED AT ": PRINT A$;SL: LPRINT A$;SL
1410 A$="INPUT:            OUTPUT:": PRINT A$: LPRINT A$
1412 ZI=0
1415 IF HL=70000 GOTO 1520
1417 IF HL<0 THEN HL=HL+65536
1418 IF H1<0 THEN H1=H1+65536
1420 A$="HL=": PRINT A$;HL,A$;H1: LPRINT A$;HL,A$;H1
1430 IF PB=70000 GOTO 1480
1440 A$="PARAM": ZA=PB
1460 GOSUB 12000
1480 IF M1=70000 GOTO 1520
1485 A$="MEMB1": ZA=M1
1490 GOSUB 12000
1500 IF M2=70000 GOTO 1520
1505 A$="MEMB2": ZA=M2
1510 GOSUB 12000
1520 GOTO 1010
10000 'SUBROUTINE TO INPUT, LIST, PRINT, AND STORE VALUES
10005 'ENTER WITH ZA=MEMORY BLOCK START
```

```
10008 ZN=ZA
10010 PRINT"+";ZN-ZA;:LPRINT "+";ZN-ZA;:INPUT ZL,ZV: LPRINT ZL;ZV
10020 IF ZL=0 GOTO 10060
10030 POKE ZN,ZV-INT(ZV/256)*256: IO(ZI)=ZV-INT(ZV/256)*256
10040 IF ZL=2 THEN POKE ZN+1,INT(ZV/256): IO(ZI+1)=INT(ZV/256)
10050 ZN=ZN+ZL: ZI=ZI+ZL
10055 GOTO 10010
10060 IO(ZI)=-1: ZI=ZI+1
10070 RETURN
12000 'SUBROUTINE TO OUTPUT VALUES FROM PARAMETER BLOCK
12010 'OR MEMORY BLOCK
12020 'ENTER WITH A$=TITLE, ZA=BLOCK START, ZI=IO() INDEX
12030 ZN=0
12040 ZB=IO(ZI): IF ZB=-1 GOTO 12090
12045 IF ZN<10 THEN ZN$=STR$(ZN)+" " ELSE ZN$=STR$(ZN)
12050 PRINT A$;"+";ZN$;ZB,A$;"+";ZN$;PEEK(ZA+ZN)
12060 LPRINT A$;"+";ZN$;ZB,A$;"+";ZN$;PEEK(ZA+ZN)
12070 ZN=ZN+1: ZI=ZI+1: GOTO 12040
12090 ZI=ZI+1: RETURN
```

What to Do if You Have Trouble

Every effort has been made to thoroughly check out and debug the subroutines in this book. If you find errors, follow this procedure:

1. If you are not using the subroutines exactly as listed, please thoroughly check out your modifications. We simply can't be responsible for your changes—there's too much chance for error. We will be responsible, however, for use of the subroutine exactly as listed in the book.

2. Verify that the subroutine checksums to the proper value as shown in the description. To do this, use the CHKSUM subroutine in the book, and checksum the subroutine in question from start to end address. The checksum must compare to that given in the book. If it does not, you have entered the data incorrectly.

3. Verify that the calling sequence and parameter values are proper. List the parameters directly before the call and see that they are within the limits imposed by the subroutine. If they are not, the subroutine may indeed not work properly or may cause the system to crash. We can't be responsible for these cases.

4. If you have done all of the above and feel there is still an error in the subroutine, then fill out the following reporting form and send it to the author at:

P.O. Box 3568
Mission Viejo, CA 92692

Your time and trouble are appreciated and the problem will be corrected for the next edition of this book.

Source Programs on Disk

A set of diskettes containing all source programs is available from the author. For information, please send a self-addressed, stamped envelope to the above address.

TRS-80 Assembly-Language Subroutines
Error Reporting Form

1. Subroutine name:

2. I am using the identical code as shown in the book: Yes No

3. I have checksummed the data: Yes No

4. Location of subroutine in memory:

5. I am using the subroutine embedded in BASIC: Yes No

6. I am using the subroutine as a stand-alone program (not embedded in BASIC): Yes No

7. System: Model I Model II Model III

8. Operating system:

9. Assembler (if applicable):

10. Input parameters:

11. Output parameters:

12. Complete description of error (please attach BASIC listing, assembly listing, or any other data you find pertinent):

13. Name:

14. Address:

Thanks for your time and trouble!

Mail to: William Barden Jr., P.O. Box 3568, Mission Viejo, CA 92692

II
TRS-80 ASSEMBLY-LANGUAGE SUBROUTINES

ABXBIN: ASCII BINARY TO BINARY CONVERSION

System Configuration

Model I, Model III, Model II Stand Alone.

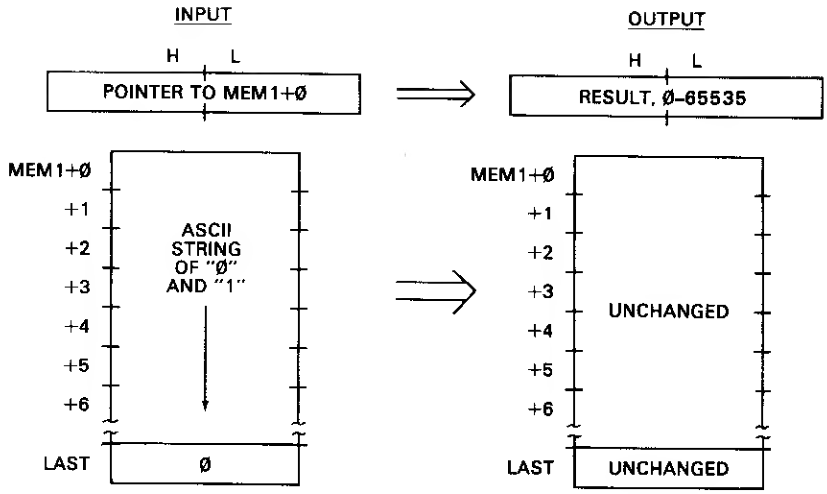
Description

ABXBIN converts a string of ASCII characters representing ones and zeroes to a 16-bit binary number. Each character in the string is assumed to be either an ASCII one (30H) or an ASCII zero (31H). The string may be from zero to 16 bytes long, but is terminated with a byte of all zeroes.

Input/Output Parameters

On input, the HL register pair contains a pointer to the string of characters.

On output, HL contains the binary number of 0 through 65,535.



Algorithm

A result of 0000000000000000 is first cleared in the IX register.

Each character is read from the string, moving from left to right. The character is first tested for a null, which marks the end of the string. If a null is found, the conversion is over.

If the character is not a null, it is assumed to be either an ASCII zero (30H) or one (31H). A value of 30H is subtracted from the character to yield a binary value of 00000000 or 00000001. This value is then added to the result in IX. Effectively, this merges the current 0 or 1 bit into the least significant bit position of the IX register. As the IX register is added to itself to cause a "shift left" one bit position at the start of each iteration of the loop, successive 0 and 1 bits move toward the left of the result. The value in IX at the end of the string represents the converted binary value.

Note that the shift is done after the test for null; this ensures that the last binary 0 or 1 remains in the least significant bit of IX.

If the ASCII string was 30H, 31H, 31H, 30H, 31H, 00H, the result in IX would be 0000000000001101.

Sample Calling Sequence

```
NAME OF SUBROUTINE? ABXBIN
HL VALUE? 40000
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION? 40000
MEMORY BLOCK 1 VALUES?
+ 0  1  49 ┐
+ 1  1  49 │
+ 2  1  49 │
+ 3  1  48 ├ 111011 IN ASCII
+ 4  1  49 │
+ 5  1  49 ┘
+ 6  1  0  TERMINATOR
```

```
+ 7   0   0
MEMORY BLOCK 2 LOCATION?
MOVE SUBROUTINE TO? 38000
SUBROUTINE EXECUTED AT   38000
INPUT:                  OUTPUT:
HL= 40000               HL= 59  RESULT
MEMB1+ 0   49           MEMB1+ 0   49
MEMB1+ 1   49           MEMB1+ 1   49
MEMB1+ 2   49           MEMB1+ 2   49
MEMB1+ 3   48           MEMB1+ 3   48   UNCHANGED
MEMB1+ 4   49           MEMB1+ 4   49
MEMB1+ 5   49           MEMB1+ 5   49
MEMB1+ 6   0            MEMB1+ 6   0

NAME OF SUBROUTINE?
```

Notes

1. If the string of ASCII characters is longer than 16 bytes, ABXBIN will return a result that represents the last 16 characters of the string.

2. If any character in the string is not a 30H or 31H, ABXBIN will return an invalid result; no check is made of the validity of the ASCII characters.

Program Listing

```
7F00             00100          ORG     7F00H           ;0522
                 00110 ;*****************************************************
                 00120 ;* ASCII BINARY TO BINARY CONVERSION. CONVERTS A STRING *
                 00130 ;* OF ASCII CHARACTERS REPRESENTING ZEROES AND ONES TO   *
                 00140 ;* BINARY.                                              *
                 00150 ;*    INPUT: HL=> STRING OF CHARACTERS, TERMINATED BY   *
                 00160 ;*           NULL CHARACTER.                            *
                 00170 ;*    OUTPUT:HL=BINARY NUMBER FROM 0 - 65535            *
                 00180 ;*****************************************************
                 00190 ;
7F00 F5          00200 ABXBIN   PUSH    AF              ;SAVE REGISTERS
7F01 D5          00210          PUSH    DE
7F02 DDE5        00220          PUSH    IX
7F04 CD7F0A      00230          CALL    0A7FH           ;***GET STRING LOC'N***
7F07 DD210000    00240          LD      IX,0            ;CLEAR RESULT REGISTER
7F0B 1600        00250          LD      D,0             ;FOR LOOP
7F0D 7E          00260 ABX010   LD      A,(HL)            ;GET NEXT ASCII CHAR
7F0E B7          00270          OR      A                 ;TEST FOR NULL (END)
7F0F 280A        00280          JR      Z,ABX020          ;GO IF END
7F11 DD29        00290          ADD     IX,IX             ;SHIFT LEFT ONE
7F13 D630        00300          SUB     30H               ;CONVERT ASCII TO 0 OR 1
7F15 5F          00310          LD      E,A               ;NOW IN E
7F16 DD19        00320          ADD     IX,DE             ;MERGE WITH PREVIOUS
7F18 23          00330          INC     HL                ;POINT TO NEXT CHARACTER
7F19 18F2        00340          JR      ABX010            ;LOOP 'TIL END
7F1B DDE5        00350 ABX020   PUSH    IX              ;TRANSFER RESULT
7F1D E1          00360          POP     HL              ;RESULT NOW IN HL
7F1E DDE1        00370          POP     IX              ;RESTORE REGISTERS
7F20 D1          00380          POP     DE
7F21 F1          00390          POP     AF
7F22 C39A0A      00400          JP      0A9AH           ;***RETURN ARGUMENT***
7F25 C9          00410          RET                     ;NON-BASIC RETURN
0000             00420          END
00000 TOTAL ERRORS
```

```
ABXBIN DECIMAL VALUES


 245, 213, 221, 229, 205, 127, 10, 221, 33, 0,
 0, 22, 0, 126, 183, 40, 10, 221, 41, 214,
 48, 95, 221, 25, 35, 24, 242, 221, 229, 225,
 221, 225, 209, 241, 195, 154, 10, 201

CHKSUM= 62
```

ADEBCD: ASCII DECIMAL TO BCD CONVERSION

System Configuration

Model I, Model III, Model II Stand Alone.

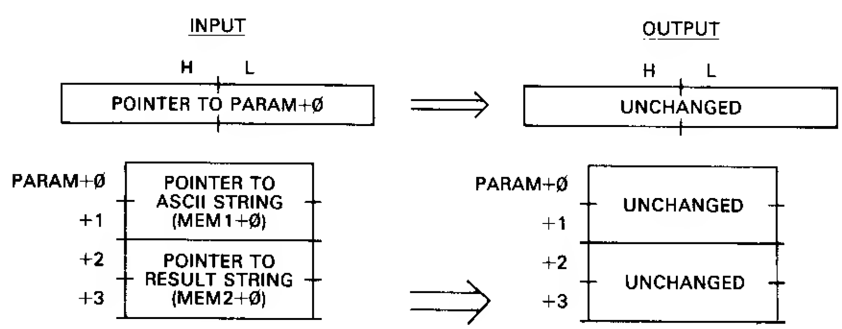
Description

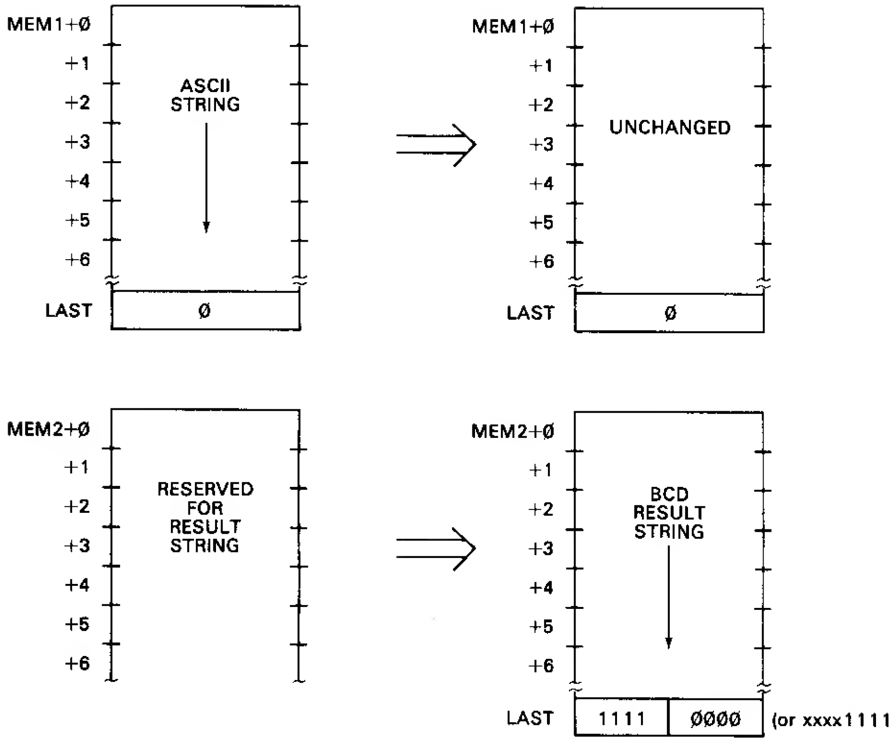
ADEBCD converts a string of ASCII characters representing ones and zeroes to a string of bcd digits. Each character in the ASCII string is assumed to be either a valid ASCII character in the range of 0 (30H) through 9 (39H). The ASCII string may be from zero to any number of bytes long, but is terminated with a byte of all zeroes. The result string of bcd digits consists of two bcd digits per byte, with a terminator of a "nibble" of ones.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the address of the ASCII string in standard Z-80 address format, least significant byte followed by most significant byte. The next two bytes of the parameter block contain the address of the result string in the same format.

On output, the parameter block and ASCII string are unchanged. The result string contains a bcd digit in one nibble (4 bits) for each byte in the ASCII string and a final nibble of ones.





Algorithm

The ADEBCD subroutine performs one conversion for each ASCII digit. The ASCII string address and result string addresses are first picked up from the parameter block and put into DE and HL, respectively.

The next ASCII character is then picked up from the ASCII string. A test is made for all zeroes. If the character is all zeroes a jump is made to ADE020.

A value of 30H is subtracted from the ASCII character to convert it to a bcd value of 0 through 9. An RLD is then done to rotate the least significant four bits of A into the result nibble. The ASCII address in DE is then incremented by one, and the next ASCII character is picked up, converted, and stored. The ASCII string pointer is again incremented to point to the next byte. The result pointer in HL is then incremented to point to the next bcd byte. A loop is then made back to ADE010.

The final action is to store all ones at the next bcd nibble position by either an RRD or RLD, depending upon the current bcd digit position.

The RRD instruction shifts the least significant four bits of the A register and the memory location pointed to by HL in a four-bit bcd shift to the right. The RLD shifts left four bits in similar fashion.

If the ASCII string was 34H, 35H, 36H, 37H, 35H, 00H, the result in the bcd string would be 45H, 67H, 5FH.

Sample Calling Sequence

```
NAME OF SUBROUTINE? ADEBCD
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0   2   47777  POINTS TO ASCII STRING
+ 2   2   48888  POINTS TO RESULT STRING
+ 4   0   0
MEMORY BLOCK 1 LOCATION? 47777
MEMORY BLOCK 1 VALUES?
+ 0   1   49 ┐
+ 1   1   57 │
+ 2   1   50 ├ 192 IN ASCII
+ 3   1   0  ┘
+ 4   0   0 TERMINATOR
MEMORY BLOCK 2 LOCATION? 48888
MEMORY BLOCK 2 VALUES?
+ 0   1   0 ┐
+ 1   1   0 ├ CLEAR RESULT FOR EXAMPLE
+ 2   0   0 ┘
MOVE SUBROUTINE TO? 45555
SUBROUTINE EXECUTED AT  45555
INPUT:               OUTPUT:
HL= 40000            HL= 40000
PARAM+ 0   161       PARAM+ 0   161 ┐
PARAM+ 1   186       PARAM+ 1   186 │
PARAM+ 2   248       PARAM+ 2   248 │
PARAM+ 3   190       PARAM+ 3   190 │
MEMB1+ 0   49        MEMB1+ 0   49  ├ UNCHANGED
MEMB1+ 1   57        MEMB1+ 1   57  │
MEMB1+ 2   50        MEMB1+ 2   50  │
MEMB1+ 3   0         MEMB1+ 3   0   ┘
MEMB2+ 0   0         MEMB2+ 0   25  ┐
MEMB2+ 1   0         MEMB2+ 1   47  ┘ 192FH = BCD 192

NAME OF SUBROUTINE?
```

Notes

1. An invalid result will occur if the ASCII string contains invalid ASCII decimal digits.

2. The terminator of all ones in the result string will be in the left-hand nibble of the result string byte (with garbage in the right-hand byte) for an even number of bcd digits, and in the right-hand nibble of the result string byte (preceded by the last bcd digit) for an odd number of bcd digits.

Program Listing

```
7F00                 ORG     7F00H           ;0522
        ;***********************************************************
        ;* ASCII DECIMAL TO BCD CONVERSION. CONVERTS A STRING     *
        ;* OF ASCII CHARACTERS REPRESENTING DECIMAL DIGITS TO     *
        ;* TO BINARY-CODED-DECIMAL.                               *
        ;*      INPUT: HL=> PARAMETER BLOCK                       *
        ;*                  PARAM+0,+1=LOCATION OF STRING OF CHARS, *
        ;*                  TERMINATED BY NULL CHARACTER          *
        ;*                  PARAM+2,+3=LOCATION OF RESULT STRING  *
        ;*     OUTPUT:RESULT STRING HOLDS STRING OF BCD DIGITS,   *
        ;                   TERMINATED BY A NIBBLE OF ONES.       *
        ;***********************************************************
        ;
```

```
7F00 F5         00230 ADEBCD  PUSH    AF              ;SAVE REGISTERS
7F01 D5         00240         PUSH    DE
7F02 E5         00250         PUSH    HL
7F03 DDE5       00260         PUSH    IX
7F05 CD7F0A     00270         CALL    0A7FH           ;***GET STRING LOC'N***
7F08 E5         00280         PUSH    HL              ;TRANSFER TO IX
7F09 DDE1       00290         POP     IX
7F0B DD5E00     00300         LD      E,(IX+0)        ;PUT SOURCE PNTR IN DE
7F0E DD5601     00310         LD      D,(IX+1)
7F11 DD6E02     00320         LD      L,(IX+2)        ;PUT DEST PNTR IN HL
7F14 DD6603     00330         LD      H,(IX+3)
7F17 1A         00340 ADE010  LD      A,(DE)            ;GET NEXT CHARACTER
7F18 B7         00350         OR      A                 ;TEST FOR NULL (END)
7F19 2005       00360         JR      NZ,ADE020         ;GO IF NOT END
7F1B 3D         00370         DEC     A                 ;ZERO TO -1
7F1C ED67       00380         RRD                       ;STORE TERMINATOR
7F1E 1816       00390         JR      ADE040            ;GO TO RETURN
7F20 D630       00400 ADE020  SUB     30H               ;CONVERT TO 0-9
7F22 ED6F       00410         RLD                       ;STORE IN BUFFER
7F24 13         00420         INC     DE                ;POINT TO NEXT CHARACTER
7F25 1A         00430         LD      A,(DE)            ;GET NEXT CHARACTER
7F26 B7         00440         OR      A                 ;TEST FOR NULL (END)
7F27 2005       00450         JR      NZ,ADE030         ;GO IF NOT END
7F29 3D         00460         DEC     A                 ;ZERO TO -1
7F2A ED6F       00470         RLD                       ;STORE TERMINATOR
7F2C 1808       00480         JR      ADE040            ;GO TO RETURN
7F2E D630       00490 ADE030  SUB     30H               ;CONVERT TO 0-9
7F30 ED6F       00500         RLD                       ;STORE IN BUFFER
7F32 13         00510         INC     DE                ;POINT TO NEXT CHARACTER
7F33 23         00520         INC     HL                ;LOC'N FOR NXT 2 BCD DGTS
7F34 18E1       00530         JR      ADE010            ;LOOP 'TIL END
7F36 DDE1       00540 ADE040  POP     IX              ;RESTORE REGISTERS
7F38 E1         00550         POP     HL
7F39 D1         00560         POP     DE
7F3A F1         00570         POP     AF
7F3B C9         00580         RET                     ;RETURN TO CALLING PROG
0000            00590         END
00000 TOTAL ERRORS
```

```
ADEBCD DECIMAL VALUES

245, 213, 229, 221, 229, 205, 127, 10, 229, 221,
225, 221, 94, 0, 221, 86, 1, 221, 110, 2,
221, 102, 3, 26, 183, 32, 5, 61, 237, 103,
24, 22, 214, 48, 237, 111, 19, 26, 183, 32,
5, 61, 237, 111, 24, 8, 214, 48, 237, 111,
19, 35, 24, 225, 221, 225, 225, 209, 241, 201

CHKSUM= 0
```

ADXBIN: ASCII DECIMAL TO BINARY CONVERSION

System Configuration

Model I, Model III, Model II Stand Alone.

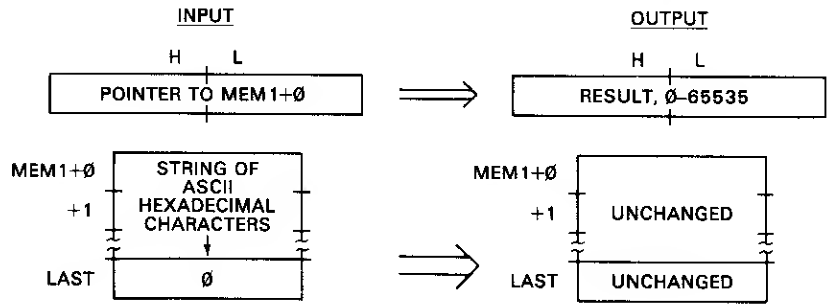
Description

ADXBIN converts a string of ASCII characters representing decimal digits to a 16-bit binary number. Each character in the string is assumed to be ASCII 0 through ASCII 9 (30H through 39H). The string may be from zero to 5 bytes long, but is terminated with a byte of all zeroes. The value represented by the string may be as large as 65,535. This conversion is an "unsigned" conversion producing a result of 0 through 65,535.

Input/Output Parameters

On input, the HL register pair contains a pointer to the string of characters.

On output, HL contains the binary number of 0 through 65,535.



Algorithm

A result of 0000000000000000 is first cleared in the IX register.

Each character is read from the string, moving from left to right. The character is first tested for a null, which marks the end of the string. If a null is found, the conversion is over.

If the character is not a null, it is assumed to be a valid ASCII decimal digit of 30H through 39H. A value of 30H is subtracted from the character to yield a binary value of 00000000 through 00001001. This value is then added to the result in IX.

Prior to the add, the partial result in the IX register is multiplied by ten. This moved the partial result over one decimal digit position to the left. The value in IX at the end of the string represents the converted binary value.

Note that the multiplication is done after the test for null; this ensures that the last value of 0 through 9 remains in the least significant decimal digit position of IX.

The multiply is done by a "shift and add" technique of three adds to shift three bits (multiply by eight) plus one add of the "times two" shift for a "times ten" result.

If the ASCII string is 34H, 35H, 30H, 31H, 31H, 00H, the result in IX would be 1010111111010011.

Sample Calling Sequence

```
NAME OF SUBROUTINE? ADXBIN
HL VALUE? 40000
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION? 40000
MEMORY BLOCK 1 VALUES?
+ 0  1  49 ┐
+ 1  1  50 │
+ 2  1  51 ├ 12345 IN ASCII
+ 3  1  52 │
+ 4  1  53 ┘
+ 5  1  0 TERMINATOR
+ 6  0  0
MEMORY BLOCK 2 LOCATION?
MOVE SUBROUTINE TO? 37000
SUBROUTINE EXECUTED AT  37000
INPUT:              OUTPUT:
HL= 40000           HL= 12345  RESULT
MEMB1+ 0   49       MEMB1+ 0   49 ┐
MEMB1+ 1   50       MEMB1+ 1   50 │
MEMB1+ 2   51       MEMB1+ 2   51 ├ UNCHANGED
MEMB1+ 3   52       MEMB1+ 3   52 │
MEMB1+ 4   53       MEMB1+ 4   53 │
MEMB1+ 5   0        MEMB1+ 5   0  ┘

NAME OF SUBROUTINE?
```

Notes

1. If the string of ASCII characters is longer than 5 bytes, or if the value represented is greater than 65,535, ADXBIN will return an invalid result.

2. If one or more characters in the string are not valid ASCII decimal digits of 30H through 39H, ADXBIN will return an invalid result; no check is made of the validity of the ASCII characters.

Program Listing

```
7F00                          ORG     7F00H           ;0522
                 ;*******************************************************
                 ;* ASCII DECIMAL TO BINARY CONVERSION. CONVERTS A STRING*
                 ;* OF ASCII CHARACTERS REPRESENTING DECIMAL DIGITS TO   *
                 ;* BINARY.                                              *
                 ;*    INPUT: HL=> STRING OF CHARACTERS, TERMINATED BY   *
                 ;*           NULL CHARACTER.                            *
                 ;*    OUTPUT:HL=BINARY NUMBER FROM 0 - 65535            *
                 ;*******************************************************
                 ;
7F00 F5          ADXBIN  PUSH    AF              ;SAVE REGISTERS
7F01 D5                  PUSH    DE
7F02 DDE5                PUSH    IX
7F04 CD7F0A              CALL    0A7FH           ;***GET STRING LOC'N***
7F07 DD210000            LD      IX,0            ;CLEAR RESULT REGISTER
7F0B 7E          ADX010  LD      A,(HL)          ;GET NEXT CHARACTER
7F0C B7                  OR      A               ;TEST FOR NULL (END)
7F0D 2815                JR      Z,ADX020        ;GO IF END
7F0F DD29                ADD     IX,IX           ;RESULT TIMES TWO
7F11 DDE5                PUSH    IX              ;SAVE RESULT
```

```
7F13 DD29      00300          ADD    IX,IX         ;RESULT TIMES FOUR
7F15 DD29      00310          ADD    IX,IX         ;RESULT TIMES EIGHT
7F17 D1        00320          POP    DE            ;GET RESULT TIMES TWO
7F18 DD19      00330          ADD    IX,DE         ;RESULT TIMES TEN
7F1A D630      00340          SUB    30H           ;CONVERT TO 0 - 9
7F1C 5F        00350          LD     E,A           ;NOW IN E
7F1D 1600      00360          LD     D,0           ;NOW IN DE
7F1F DD19      00370          ADD    IX,DE         ;MERGE WITH PREVIOUS
7F21 23        00380          INC    HL            ;POINT TO NEXT CHARACTER
7F22 18E7      00390          JR     ADX010        ;LOOP 'TIL END
7F24 DDE5      00400 ADX020   PUSH   IX          ;TRANSFER RESULT
7F26 E1        00410          POP    HL          ;RESULT NOW IN HL
7F27 DDE1      00420          POP    IX          ;RESTORE REGISTERS
7F29 D1        00430          POP    DE
7F2A F1        00440          POP    AF
7F2B C39A0A    00450          JP     0A9AH       ;***RETURN ARGUMENT***
7F2E C9        00460          RET                ;NON-BASIC RETURN
0000           00470          END
00000 TOTAL ERRORS
```

ADXBIN DECIMAL VALUES

```
245, 213, 221, 229, 205, 127, 10, 221, 33, 0,
0, 126, 183, 40, 21, 221, 41, 221, 229, 221,
41, 221, 41, 209, 221, 25, 214, 48, 95, 22,
0, 221, 25, 35, 24, 231, 221, 229, 225, 221,
225, 209, 241, 195, 154, 10, 201
```

CHKSUM= 211

AHXBIN: ASCII HEXADECIMAL TO BINARY CONVERSION

System Configuration

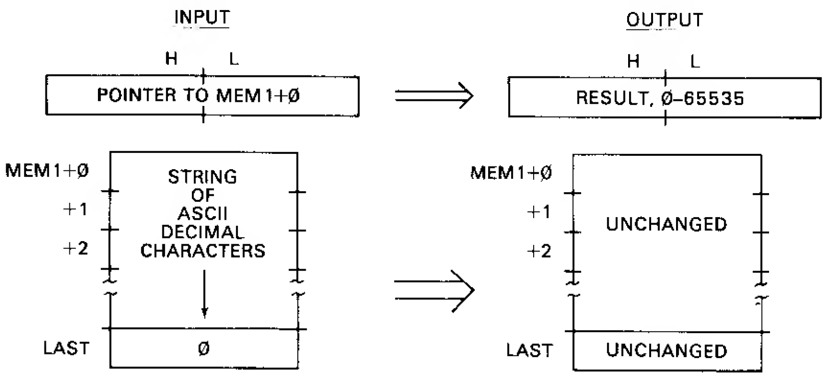
Model I, Model III, Model II Stand Alone.

Description

AHXBIN converts a string of ASCII characters representing hexadecimal digits to a 16-bit binary number. Each character in the string is assumed to be either in the range of ASCII 0 through 7 (30H through 37H) or ASCII A through F (41H through 46H). The string may be from zero to 4 bytes long, but is terminated with a byte of all zeroes.

Input/Output Parameters

On input, the HL register pair contains a pointer to the string of characters.



On output, HL contains the binary number of 0 through 65,535.

Algorithm

A result of 0000000000000000 is first cleared in the IX register.

Each character is read from the string, moving from left to right. The character is first tested for a null, which marks the end of the string. If a null is found, the conversion is over.

If the character is not a null, it is assumed to be in the proper range for hexadecimal digits. A value of 30H is subtracted from the character to yield a value of 0 through 9 or 17 through 22. This value is then tested for the second set of values of 17 through 22 by subtracting 10. If the original value was 0 through 9, the result of this subtract will be negative, and the original value of 0 through 9 is used. If the result was positive, the value is now 7 through 12, and is changed to the proper hex value by adding 3, to produce 10 through 15. This value is then added to the result in IX. Effectively, this merges the four bits of the current value into the four least significant bit positions of the IX register.

As the IX register is added to itself four times to cause a "shift left" four bit positions at the start of each iteration of the loop, successive hex digits move toward the left of the result. The value in IX at the end of the string represents the converted binary value.

Note that the shifts are done after the test for null; this ensures that the last octal digit remains in the least significant four bits of IX.

If the ASCII string was 41H, 45H, 31H, and 00H, the result in IX would be 0000101011100001, or hex 0AE1.

Sample Calling Sequence

```
NAME OF SUBROUTINE? AHXBIN
HL VALUE? 50000
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION? 50000
MEMORY BLOCK 1 VALUES?
+ 0  1  70 ┐
+ 1  1  49 ├ FIA9 IN ASCII
+ 2  1  65 │
+ 3  1  57 ┘
+ 4  1  0  TERMINATOR
+ 5  0  0
MEMORY BLOCK 2 LOCATION?
MOVE SUBROUTINE TO? 40000
SUBROUTINE EXECUTED AT  40000
INPUT:               OUTPUT:
HL= 50000            HL= 61865 RESULT = FIA9H
MEMB1+ 0  70         MEMB1+ 0  70 ┐
MEMB1+ 1  49         MEMB1+ 1  49 │
MEMB1+ 2  65         MEMB1+ 2  65 ├ UNCHANGED
MEMB1+ 3  57         MEMB1+ 3  57 │
MEMB1+ 4  0          MEMB1+ 4  0  ┘

NAME OF SUBROUTINE?
```

Notes

1. If the string of ASCII characters is longer than 4 bytes, AHXBIN will return a result that represents the last 4 characters of the string.

2. If any character in the string is not in the proper range, AHXBIN will return an invalid result; no check is made of the validity of the ASCII characters.

Program Listing

```
7F00            00100           ORG     7F00H           ;0522
                00110 ;**********************************************************
                00120 ;* ASCII HEXADECIMAL TO BINARY CONVERSION. CONVERTS A    *
                00130 ;* STRING OF ASCII CHARACTERS REPRESENTING HEXADECIMAL   *
                00140 ;* DIGITS TO BINARY.                                     *
                00150 ;*     INPUT: HL=> STRING OF CHARACTERS, TERMINATED BY    *
                00160 ;*            NULL CHARACTER.                             *
                00170 ;*     OUTPUT:HL=BINARY NUMBER FROM 0 - 65535             *
                00180 ;**********************************************************
                00190 ;
7F00 F5         00200 AHXBIN    PUSH    AF              ;SAVE REGISTERS
7F01 D5         00210           PUSH    DE
7F02 DDE5       00220           PUSH    IX
7F04 CD7F0A     00230           CALL    0A7FH           ;***GET STRING LOC'N***
7F07 DD210000   00240           LD      IX,0            ;CLEAR RESULT REGISTER
7F0B 1600       00250           LD      D,0             ;FOR LOOP
7F0D 7E         00260 AHX010    LD      A,(HL)            ;GET NEXT CHARACTER
7F0E B7         00270           OR      A                 ;TEST FOR NULL (END)
7F0F 2819       00280           JR      Z,AHX020          ;GO IF END
7F11 DD29       00290           ADD     IX,IX             ;SHIFT LEFT 4 BITS
7F13 DD29       00300           ADD     IX,IX
7F15 DD29       00310           ADD     IX,IX
7F17 DD29       00320           ADD     IX,IX
7F19 D630       00330           SUB     30H               ;CONVERT TO 0-9 OR 11-16
7F1B 5F         00340           LD      E,A               ;NOW IN E
7F1C D60A       00350           SUB     0AH               ;SUBTRACT FOR A - F
7F1E CB7F       00360           BIT     7,A               ;TEST RESULT
7F20 2003       00370           JR      NZ,AHX015         ;GO IF 0 - 9
7F22 C603       00380           ADD     A,3               ;CONVERT TO A - F
7F24 5F         00390           LD      E,A               ;NOW IN E
7F25 DD19       00400 AHX015    ADD     IX,DE             ;MERGE WITH PREVIOUS
7F27 23         00410           INC     HL                ;POINT TO NEXT CHARACTER
7F28 18E3       00420           JR      AHX010            ;LOOP 'TIL END
7F2A DDE5       00430 AHX020    PUSH    IX              ;TRANSFER RESULT
7F2C E1         00440           POP     HL
7F2D DDE1       00450           POP     IX              ;RESTORE REGISTERS
7F2F D1         00460           POP     DE
7F30 F1         00470           POP     AF
7F31 C39A0A     00480           JP      0A9AH           ;***RETURN ARGUMENT***
7F34 C9         00490           RET                     ;NON-BASIC RETURN
0000            00500           END
00000 TOTAL ERRORS
```

```
AHXBIN DECIMAL VALUES


 245, 213, 221, 229, 205, 127, 10, 221, 33, 0,
 0, 22, 0, 126, 183, 40, 25, 221, 41, 221,
 41, 221, 41, 221, 41, 214, 48, 95, 214, 10,
 203, 127, 32, 3, 198, 3, 95, 221, 25, 35,
 24, 227, 221, 229, 225, 221, 225, 209, 241, 195,
 154, 10, 201


CHKSUM= 197
```

AOXBIN: ASCII OCTAL TO BINARY CONVERSION

System Configuration

Model I, Model III, Model II Stand Alone.

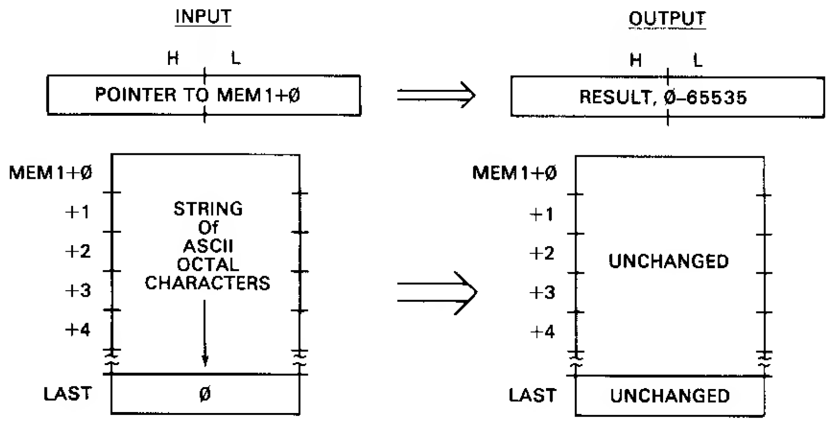
Description

AOXBIN converts a string of ASCII characters representing octal digits to a 16-bit binary number. Each character in the string is assumed to be in the range of ASCII 0 through 7 (30H through 37H). The string may be from zero to 6 bytes long, but is terminated with a byte of all zeroes.

Input/Output Parameters

On input, the HL register pair contains a pointer to the string of characters.

On output, HL contains the binary number of 0 through 65,535.



Algorithm

A result of 0000000000000000 is first cleared in the IX register.

Each character is read from the string, moving from left to right. The character is first tested for a null, which marks the end of the string. If a null is found, the conversion is over.

If the character is not a null, it is assumed to be in the proper range for octal digits. A value of 30H is subtracted from the character to yield a value of 0 through 7. This value is then added to the result in IX. Effectively, this merges the three bits of the current value into the three least significant bit positions of the IX register.

As the IX register is added to itself three times to cause a "shift left" three bit positions at the start of each iteration of the loop, successive octal digits move toward the left of the result. The value in IX at the end of the string represents the converted binary value.

Note that the shifts are done after the test for null; this ensures that the last octal digit remains in the least significant three bits of IX.

If the ASCII string was 33H, 37H, 35H, and 00H, the result in IX would be 0000000011111101, or octal 375.

Sample Calling Sequence

```
NAME OF SUBROUTINE? AOXBIN
HL VALUE? 40000
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION? 40000
MEMORY BLOCK 1 VALUES?
+ 0   1   49 ┐
+ 1   1   50 │
+ 2   1   51 ├ 123457 IN ASCII
+ 3   1   52 │
+ 4   1   53 │
+ 5   1   55 ┘
+ 6   1   0 TERMINATOR
+ 7   0   0
MEMORY BLOCK 2 LOCATION?
MOVE SUBROUTINE TO? 37000
SUBROUTINE EXECUTED AT  37000
INPUT:                OUTPUT:
HL= 40000             HL= 42799 RESULT
MEMB1+ 0   49         MEMB1+ 0   49 ┐
MEMB1+ 1   50         MEMB1+ 1   50 │
MEMB1+ 2   51         MEMB1+ 2   51 │
MEMB1+ 3   52         MEMB1+ 3   52 ├ UNCHANGED
MEMB1+ 4   53         MEMB1+ 4   53 │
MEMB1+ 5   55         MEMB1+ 5   55 │
MEMB1+ 6   0          MEMB1+ 6   0  ┘

NAME OF SUBROUTINE?
```

Notes

1. If the string of ASCII characters is longer than 6 bytes, or if the octal value represented is greater than 177777, AOXBIN will return an invalid result.

2. If any character in the string is not in the proper range, AOXBIN will return an invalid result; no check is made of the validity of the ASCII characters.

Program Listing

```
7F00            00100         ORG     7F00H           ;0522
                00110 ;*****************************************************
                00120 ;* ASCII OCTAL TO BINARY CONVERSION. CONVERTS A STRING  *
                00130 ;* OF ASCII CHARACTERS REPRESENTING OCTAL DIGITS TO BI- *
                00140 ;* NARY.                                                *
                00150 ;*    INPUT: HL=> STRING OF CHARACTERS, TERMINATED BY   *
                00160 ;*           NULL CHARACTER.                            *
                00170 ;*    OUTPUT:HL=BINARY NUMBER FROM 0 - 65535            *
                00180 ;*****************************************************
                00190 ;
7F00 F5         00200 AOXBIN  PUSH    AF              ;SAVE REGISTERS
7F01 D5         00210         PUSH    DE
7F02 DDE5       00220         PUSH    IX
7F04 CD7F0A     00230         CALL    0A7FH           ;***GET STRING LOC'N***
```

```
7F07 DD210000 00240          LD      IX,0            ;CLEAR RESULT REGISTER
7F0B 1600     00250          LD      D,0             ;FOR LOOP
7F0D 7E       00260 AOX010   LD      A,(HL)            ;GET NEXT CHARACTER
7F0E B7       00270          OR      A                 ;TEST FOR NULL (END)
7F0F 280E     00280          JR      Z,AOX020          ;GO IF END
7F11 DD29     00290          ADD     IX,IX             ;SHIFT LEFT 3 BITS
7F13 DD29     00300          ADD     IX,IX
7F15 DD29     00310          ADD     IX,IX
7F17 D630     00320          SUB     30H               ;CONVERT TO 0-7
7F19 5F       00330          LD      E,A               ;NOW IN E
7F1A DD19     00340 AOX015   ADD     IX,DE             ;MERGE WITH PREVIOUS
7F1C 23       00350          INC     HL                ;POINT TO NEXT CHARACTER
7F1D 18EE     00360          JR      AOX010            ;LOOP 'TIL END
7F1F DDE5     00370 AOX020   PUSH    IX              ;TRANSFER RESULT
7F21 E1       00380          POP     HL
7F22 DDE1     00390          POP     IX              ;RESTORE REGISTERS
7F24 D1       00400          POP     DE
7F25 F1       00410          POP     AF
7F26 C39A0A   00420          JP      0A9AH           ;***RETURN ARGUMENT***
7F29 C9       00430          RET                     ;NON-BASIC RETURN
0000          00440          END
00000 TOTAL ERRORS
```

```
AOXBIN DECIMAL VALUES

245, 213, 221, 229, 205, 127, 10, 221, 33, 0,
0, 22, 0, 126, 183, 40, 14, 221, 41, 221,
41, 221, 41, 214, 48, 95, 221, 25, 35, 24,
238, 221, 229, 225, 221, 225, 209, 241, 195, 154,
10, 201

CHKSUM= 74
```

BCADDN: MULTIPLE-PRECISION BCD ADD

System Configuration

Model I, Model III, Model II Stand Alone.

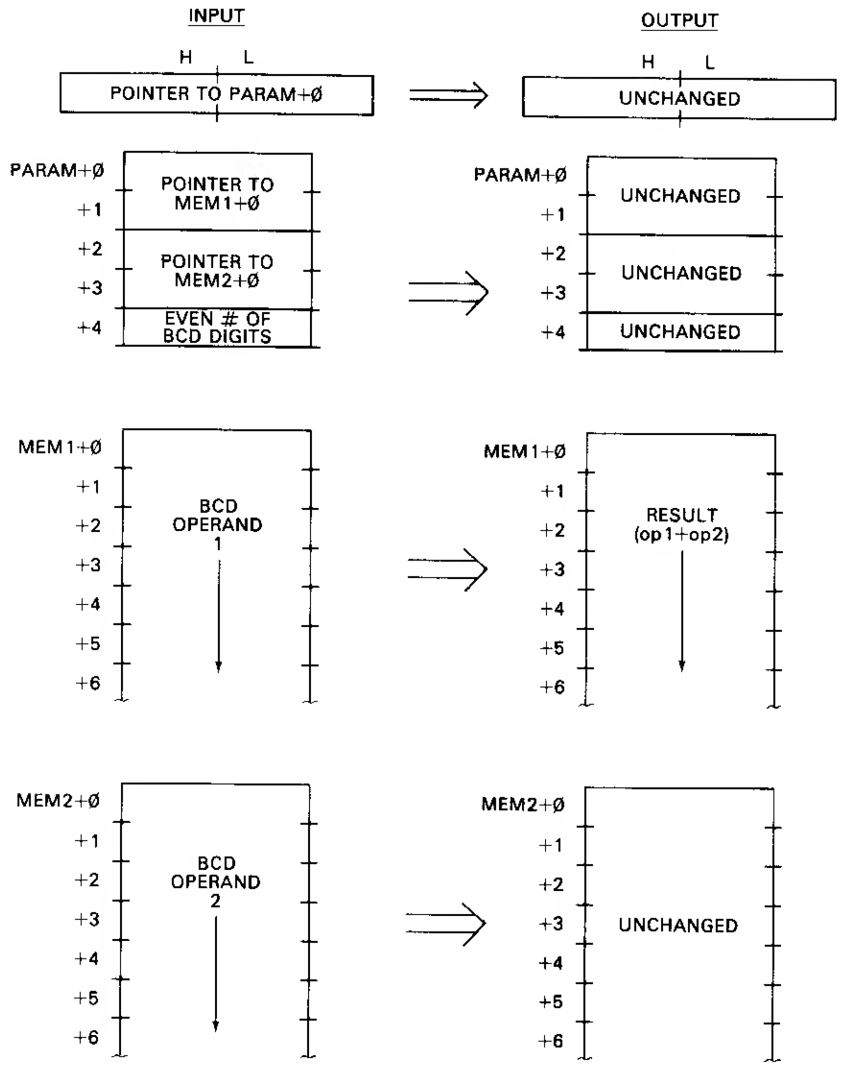
Description

BCADDN adds a "source" string of bcd digits to a "destination" string of bcd digits and puts the result of the add into the destination string. Each of the two strings is assumed to be the same length. The length must be an even number of bcd digits, but may be any number from 2 through 254.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the address of the destination string in standard Z-80 address format, least significant byte followed by most significant byte. The next two bytes of the parameter block contain the address of the source string in the same format. The next byte of the parameter block-contains the number of bcd digits in the two operands. This must be an even number (an integral number of bytes).

On output, the parameter block and source string are unchanged. The destination string contains the result of the bcd add.



Algorithm

The BCADDN subroutine performs one add for each two bcd digits. The destination string address and source string address are first picked up from the parameter block and put into DE and HL, respectively. The number of bytes in the add is then picked up and put into the BC register pair. This number is divided by two to obtain the total number of bytes involved. This number minus one is then added to the source and destination pointers so that they point to the least significant bytes of the source and destination strings. The number of bytes is then put into the B register for loop control.

The next two bcd destination digits are then picked up from the destination string (DE register pointer). An ADC is made of the two source string digits (HL register pointer). The result is adjusted for a bcd add by a DAA instruction, and the result stored in the destination string.

The source and destination string pointers are then decremented by one to point to the next most significant two bcd digits of each operand. The B register count is then decremented by a DJNZ, and a loop back to BCA010 is made for the next add.

The carry is cleared before the first bcd add, but successive adds add in the carry from the preceding bcd add.

If the destination operand was 00H, 45H, 67H, 11H and the source operand was 00H, 75H, 77H, 33H, then the number of bcd digits must be 8. The result in the destination operand would be 01H, 21H, 44H, 44H. Note that the result may be one bcd digit longer than the original number of bcd digits.

Sample Calling Sequence

```
NAME OF SUBROUTINE? BCADDN
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0  2  45000
+ 2  2  50000
+ 4  1  6   6 BCD DIGITS
+ 5  0  0
MEMORY BLOCK 1 LOCATION? 45000
MEMORY BLOCK 1 VALUES?
+ 0  1  18 ┐
+ 1  1  52 ├ 123456 IN BCD
+ 2  1  86 ┘
+ 3  0  0
MEMORY BLOCK 2 LOCATION? 50000
MEMORY BLOCK 2 VALUES?
+ 0  1  119┐
+ 1  1  5  ├ 770547 IN BCD
+ 2  1  71 ┘
+ 3  0  0
MOVE SUBROUTINE TO? 37000
SUBROUTINE EXECUTED AT  37000
INPUT:                OUTPUT:
HL= 40000             HL= 40000
PARAM+ 0  200         PARAM+ 0  200┐
PARAM+ 1  175         PARAM+ 1  175│
PARAM+ 2  80          PARAM+ 2  80 ├ UNCHANGED
PARAM+ 3  195         PARAM+ 3  195│
PARAM+ 4  6           PARAM+ 4  6  ┘
MEMB1+ 0  18          MEMB1+ 0  137┐
MEMB1+ 1  52          MEMB1+ 1  64 ├ 894003 RESULT IN BCD
MEMB1+ 2  86          MEMB1+ 2  3  ┘
MEMB2+ 0  119         MEMB2+ 0  119┐
MEMB2+ 1  5           MEMB2+ 1  5  ├ UNCHANGED
MEMB2+ 2  71          MEMB2+ 2  71 ┘

NAME OF SUBROUTINE?
```

Notes

1. An invalid result will occur if the source or destination strings do not contain valid bcd digits.

2. The destination string is a fixed length. Leading zero bcd digits must precede the operands to handle the result, which may be one bcd digit larger than either of the operands.

3. This is an "unsigned" bcd add. Both operands are assumed to be positive bcd numbers.

Program Listing

```
7F00           00100          ORG     7F00H           ;0522
               00110 ;*******************************************************
               00120 ;* MULTIPLE-PRECISION BCD ADD. ADDS TWO MULTIPLE-PRE-   *
               00130 ;* CISION BCD OPERANDS, ANY LENGTH                      *
               00140 ;*    INPUT: HL=> PARAMETER BLOCK                       *
               00150 ;*           PARAM+0,+1=ADDRESS OF OPERAND 1            *
               00160 ;*           PARAM+2,+3=ADDRESS OF OPERAND 2            *
               00170 ;*           PARAM+4=EVEN # OF BCD DIGITS, 0-254        *
               00180 ;*    OUTPUT:OPERAND 1 LOCATION HOLDS RESULT            *
               00190 ;*******************************************************
               00200 ;
7F00 F5        00210 BCADDN   PUSH    AF              ;SAVE REGISTERS
7F01 C5        00220          PUSH    BC
7F02 D5        00230          PUSH    DE
7F03 E5        00240          PUSH    HL
7F04 DDE5      00250          PUSH    IX
7F06 CD7F0A    00260          CALL    0A7FH           ;***GET PB LOC'N***
7F09 E5        00270          PUSH    HL              ;TRANSFER TO IX
7F0A DDE1      00280          POP     IX
7F0C DD5E00    00290          LD      E,(IX+0)        ;GET OP 1 LOC'N
7F0F DD5601    00300          LD      D,(IX+1)
7F12 DD6E02    00310          LD      L,(IX+2)        ;GET OP 2 LOC'N
7F15 DD6603    00320          LD      H,(IX+3)
7F18 DD4E04    00330          LD      C,(IX+4)        ;GET # OF BYTES
7F1B CB39      00340          SRL     C               ;N/2
7F1D 0600      00350          LD      B,0             ;NOW IN BC
7F1F 0B        00360          DEC     BC              ;#-1
7F20 09        00370          ADD     HL,BC           ;POINT TO LAST OP2
7F21 EB        00380          EX      DE,HL           ;SWAP DE AND HL
7F22 09        00390          ADD     HL,BC           ;POINT TO LAST OP1
7F23 EB        00400          EX      DE,HL           ;SWAP BACK
7F24 41        00410          LD      B,C             ;#-1 BACK TO B
7F25 04        00420          INC     B               ;ORIGINAL NUMBER
7F26 B7        00430          OR      A               ;CLEAR CARRY FOR FIRST ADD
7F27 1A        00440 BCA010   LD      A,(DE)            ;GET OPERAND 1 BYTE
7F28 8E        00450          ADC     A,(HL)            ;ADD OPERAND 2
7F29 27        00460          DAA                       ;DECIMAL ADJUST
7F2A 12        00470          LD      (DE),A            ;STORE RESULT
7F2B 2B        00480          DEC     HL                ;POINT TO NEXT OP2
7F2C 1B        00490          DEC     DE                ;POINT TO NEXT OP1
7F2D 10F8      00500          DJNZ    BCA010            ;LOOP FOR N BYTES
7F2F DDE1      00510          POP     IX              ;RESTORE REGISTERS
7F31 E1        00520          POP     HL
7F32 D1        00530          POP     DE
7F33 C1        00540          POP     BC
7F34 F1        00550          POP     AF
7F35 C9        00560          RET                     ;RETURN TO CALLING PROG
0000           00570          END
00000 TOTAL ERRORS
```

```
BCADDN DECIMAL VALUES

 245, 197, 213, 229, 221, 229, 205, 127, 10, 229,
 221, 225, 221, 94, 0, 221, 86, 1, 221, 110,
 2, 221, 102, 3, 221, 78, 4, 203, 57, 6,
 0, 11, 9, 235, 9, 235, 65, 4, 183, 26,
 142, 39, 18, 43, 27, 16, 248, 221, 225, 225,
 209, 193, 241, 201

CHKSUM= 115
```

System Configuration

Model I, Model III, Model II Stand Alone.

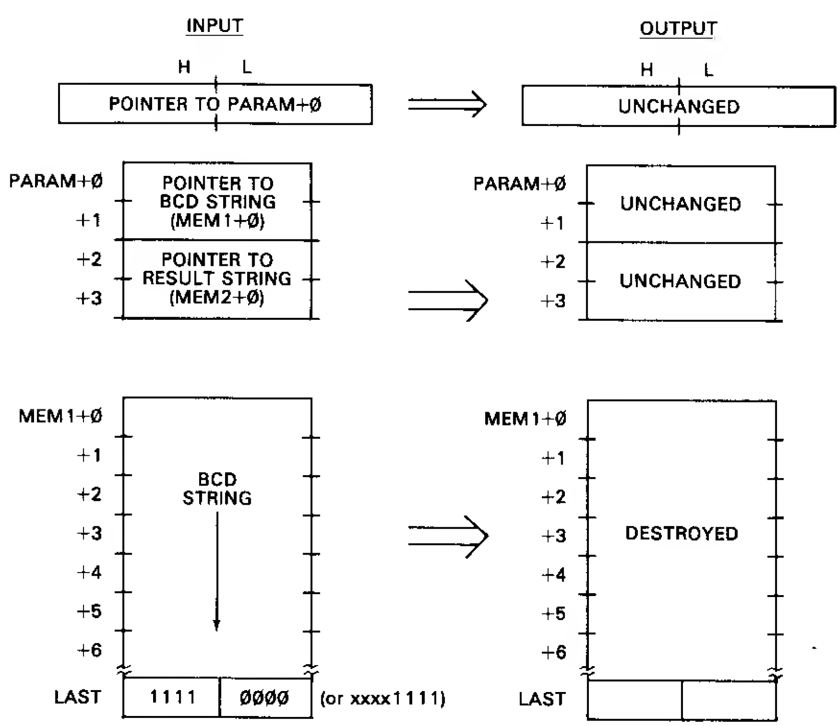
Description

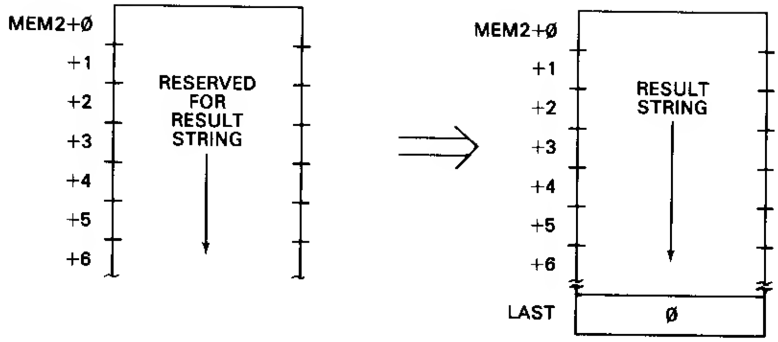
BCDXAD converts a string of bcd digits to a string of ASCII characters. Each "nibble" of four bits in the bcd string is assumed to be a valid bcd character of binary value 0 through 9. The bcd string may be from zero to any number of bytes long, but is terminated with a nibble of all ones. The result string of ASCII digits will represent ASCII decimal digits of 30H through 39H, with a terminator of a byte of zeroes.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the address of the bcd string in standard Z-80 address format, least significant byte followed by most significant byte. The next two bytes of the parameter block contain the address of the result string in the same format.

On output, the parameter block is unchanged. The bcd string is destroyed. The result string contains an ASCII decimal digit for each bcd digit in the bcd string and a final byte of zeroes.





Algorithm

The BCDXAD subroutine performs one conversion for each bcd digit. The bcd string address and result string address are first picked up from the parameter block and put into HL and DE, respectively.

The next bcd digit is then picked up from the bcd string by an RLD instruction. A test is made for all ones. If the digit is all ones, a jump is made to BCD020.

A value of 30H is added to the bcd digit to convert it to an ASCII digit of 30H through 39H. This digit is then stored in the result string. The ASCII result string address in DE is then incremented by one, and the next bcd digit is picked up, tested, converted, and stored. The ASCII string pointer is again incremented to point to the next byte. The bcd pointer in HL is then incremented to point to the next two bcd digits. A loop is then made back to BCD010.

The final action at BCD020 is to store a null (zeroes) at the next ASCII character position.

The RLD instruction shifts the least significant four bits of the A register and the memory location pointed to by HL in a four-bit bcd shift to the left.

If the bcd string was 45H, 67H, 5FH, the result in the ASCII string would be 34H, 35H, 36H, 37H, 35H, 00H.

Sample Calling Sequence

```
NAME OF SUBROUTINE? BCDXAD
HL VALUE? 41000
PARAMETER BLOCK LOCATION? 41000
PARAMETER BLOCK VALUES?
+ 0  2  44000  POINTS TO BCD STRING
+ 2  2  45000  POINTS TO RESULT STRING
+ 4  0  0
MEMORY BLOCK 1 LOCATION? 44000
MEMORY BLOCK 1 VALUES?
+ 0  1  145 ]
+ 1  1  47  ]- 912 IN BCD PLUS TERMINATOR OF ALL ONES
+ 2  0  0
MEMORY BLOCK 2 LOCATION? 45000
MEMORY BLOCK 2 VALUES?
+ 0  1  255 ]
+ 1  1  255 ]
+ 2  1  255 ]-INITIALIZE RESULT FOR EXAMPLE
+ 3  1  255 ]
+ 4  0  0
```

```
MOVE SUBROUTINE TO? 47000
SUBROUTINE EXECUTED AT  47000
INPUT:              OUTPUT:
HL= 41000           HL= 41000
PARAM+ 0   224      PARAM+ 0   224
PARAM+ 1   171      PARAM+ 1   171
PARAM+ 2   200      PARAM+ 2   200
PARAM+ 3   175      PARAM+ 3   175
MEMB1+ 0   145      MEMB1+ 0   0
MEMB1+ 1   47       MEMB1+ 1   0
MEMB2+ 0   255      MEMB2+ 0   57 ┐
MEMB2+ 1   255      MEMB2+ 1   49 ├ 912 IN ASCII
MEMB2+ 2   255      MEMB2+ 2   50 ┘
MEMB2+ 3   255      MEMB2+ 3   0 TERMINATOR

NAME OF SUBROUTINE?
```

Notes

1. An invalid result will occur if the bcd string contains invalid bcd digits.
2. The bcd string will be destroyed in the processing.

Program Listing

```
7F00            00100          ORG      7F00H            ;0522
                00110 ;*****************************************************
                00120 ;* BCD TO ASCII DECIMAL CONVERSION. CONVERTS A STRING  *
                00130 ;* OF BCD DIGITS TO A STRING OF ASCII CHARACTERS.      *
                00140 ;*     INPUT: HL=> PARAMETER BLOCK                      *
                00150 ;*            PARAM+0,+1=LOCATION OF STRING OF BCD DGTS, *
                00160 ;*            TERMINATED BY A NIBBLE OF ALL ONES.       *
                00170 ;*            PARAM+2,+3=LOCATION OF RESULT STRING      *
                00180 ;*     OUTPUT:RESULT STRING HOLDS STRING OF ASCII CHARS, *
                00190 ;             TERMINATED BY A NULL.                     *
                00200 ;*****************************************************
                00210 ;
7F00 F5         00220 BCDXAD   PUSH     AF               ;SAVE REGISTERS
7F01 D5         00230          PUSH     DE
7F02 E5         00240          PUSH     HL
7F03 DDE5       00250          PUSH     IX
7F05 CD7F0A     00260          CALL     0A7FH            ;***GET STRING LOC'N***
7F08 E5         00270          PUSH     HL               ;TRANSFER TO IX
7F09 DDE1       00280          POP      IX
7F0B DD5E02     00290          LD       E,(IX+2)         ;PUT DEST PNTR IN DE
7F0E DD5603     00300          LD       D,(IX+3)
7F11 DD6E00     00310          LD       L,(IX+0)         ;PUT SOURCE PNTR IN HL
7F14 DD6601     00320          LD       H,(IX+1)
7F17 AF         00330 BCD010   XOR      A                  ;CLEAR A
7F18 ED6F       00340          RLD                         ;GET BCD DIGIT
7F1A FE0F       00350          CP       0FH                ;TEST FOR ONES (END)
7F1C 2812       00360          JR       Z,BCD020           ;GO IF END
7F1E C630       00370          ADD      A,30H              ;CONVERT TO 0-9 ASCII
7F20 12         00380          LD       (DE),A             ;STORE ASCII CHAR
7F21 13         00390          INC      DE                 ;POINT TO NEXT CHARACTER
7F22 AF         00400          XOR      A                  ;CLEAR A
7F23 ED6F       00410          RLD                         ;GET BCD DIGIT
7F25 FE0F       00420          CP       0FH                ;TEST FOR ONES (END)
7F27 2807       00430          JR       Z,BCD020           ;GO IF END
7F29 C630       00440          ADD      A,30H              ;CONVERT TO 0-9
7F2B 12         00450          LD       (DE),A             ;STORE ASCII CHAR
7F2C 13         00460          INC      DE                 ;POINT TO NEXT CHARACTER
7F2D 23         00470          INC      HL                 ;LOC'N FOR NXT 2 BCD DGTS
7F2E 18E7       00480          JR       BCD010             ;LOOP 'TIL END
```

```
7F30 AF         00490 BCD020  XOR    A           ;NULL
7F31 12         00500         LD     (DE),A      ;STORE NULL AS TERMINATOR
7F32 DDE1       00510         POP    IX          ;RESTORE REGISTERS
7F34 E1         00520         POP    HL
7F35 D1         00530         POP    DE
7F36 F1         00540         POP    AF
7F37 C9         00550         RET                ;RETURN TO CALLING PROG
0000            00560         END
00000 TOTAL ERRORS
```

```
BCDXAD DECIMAL VALUES


 245, 213, 229, 221, 229, 205, 127, 10, 229, 221,
 225, 221, 94, 2, 221, 86, 3, 221, 110, 0,
 221, 102, 1, 175, 237, 111, 254, 15, 40, 18,
 198, 48, 18, 19, 175, 237, 111, 254, 15, 40,
 7, 198, 48, 18, 19, 35, 24, 231, 175, 18,
 221, 225, 225, 209, 241, 201

CHKSUM= 72
```

BCSUBT: MULTIPLE-PRECISION BCD SUBTRACT

System Configuration

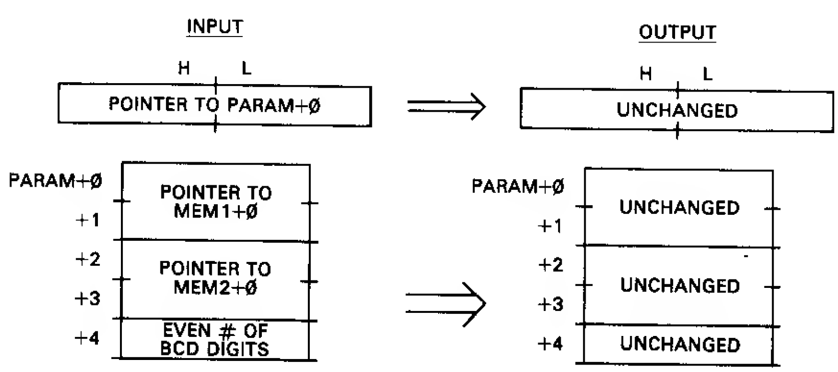
Model I, Model III, Model II Stand Alone.

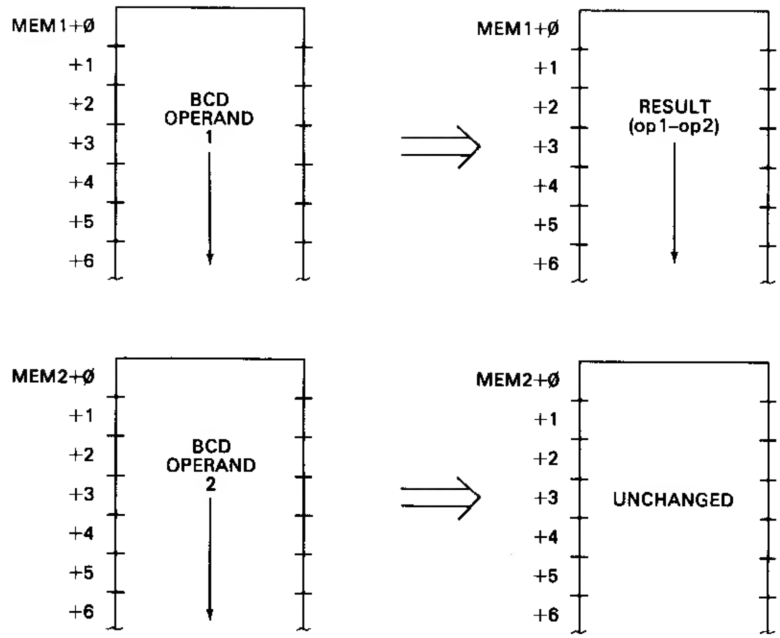
Description

BCSUBT subtracts a "source" string of bcd digits from a "destination" string of bcd digits and puts the result of the subtract into the destination string. Each of the two strings is assumed to be the same length. The length must be an even number of bcd digits, but may be any number from 2 through 254.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the address of the destination string in standard Z-80 address format, least significant byte followed by most significant byte. The next two bytes of the parameter block contain the address of the source string in the same format. The next byte of the parameter block contains the number of bcd digits in the two operands. This must be an even number (an integral number of bytes).





On output, the parameter block and source string are unchanged. The destination string contains the result of the bcd subtract.

Algorithm

The BCSUBT subroutine performs one subtract for each two bcd digits. The destination string address and source string address are first picked up from the parameter block and put into DE and HL, respectively. The number of bytes in the subtract is then picked up and put into the BC register pair. This number is divided by two to obtain the total number of bytes involved. This number minus one is then added to the source and destination pointers so that they point to the least significant bytes of the source and destination strings. The number of bytes is then put into the B register for loop control.

The next two bcd destination digits are then picked up from the destination string (DE register pointer). An ADC is made of the two source string digits (HL register pointer). The result is adjusted for a bcd subtract by a DAA instruction, and the result stored in the destination string.

The source and destination string pointers are then decremented by one to point to the next most significant two bcd digits of each operand. The B register count is then decremented by a DJNZ, and a loop back to BCS010 is made for the next subtract.

The carry is cleared before the first bcd subtract, but successive subtracts subtract in the carry from the preceding bcd subtract.

If the destination operand was 00H, 45H, 67H, 11H and the source operand was 00H, 75H, 77H, 33H, then the number of bcd digits must be 8. The result in the destination operand would be 99H, 69H, 89H, 78H.

Sample Calling Sequence

```
NAME OF SUBROUTINE? BCSUBT
HL VALUE? 50000
PARAMETER BLOCK LOCATION? 50000
PARAMETER BLOCK VALUES?
+ 0  2  52000
+ 2  2  54000
+ 4  1  4         4 BCD DIGITS
+ 5  0  0
MEMORY BLOCK 1 LOCATION? 52000
MEMORY BLOCK 1 VALUES?
+ 0  1  149 ]
+ 1  1  112 ]- 9570 IN BCD
+ 2  0  0
MEMORY BLOCK 2 LOCATION? 54000
MEMORY BLOCK 2 VALUES?
+ 0  1  147 ]
+ 1  1  131 ]- 9383 IN BCD
+ 2  0  0
MOVE SUBROUTINE TO? 45000
SUBROUTINE EXECUTED AT  45000
INPUT:              OUTPUT:
HL= 50000           HL= 50000
PARAM+ 0   32       PARAM+ 0   32  ]
PARAM+ 1   203      PARAM+ 1   203 ]
PARAM+ 2   240      PARAM+ 2   240 ]- UNCHANGED
PARAM+ 3   210      PARAM+ 3   210 ]
PARAM+ 4   4        PARAM+ 4   4   ]
MEMB1+ 0   149      MEMB1+ 0   1   ]
MEMB1+ 1   112      MEMB1+ 1   135 ]- 187 RESULT IN BCD
MEMB2+ 0   147      MEMB2+ 0   147 ]
MEMB2+ 1   131      MEMB2+ 1   131 ]-UNCHANGED

NAME OF SUBROUTINE?
```

Notes

1. An invalid result will occur if the source or destination strings do not contain valid bcd digits.

2. This is an "unsigned" subtract. Both operands are assumed to be positive bcd numbers.

Program Listing

```
7F00                      ORG     7F00H           ;0522
                ;*****************************************************
                ;* MULTIPLE-PRECISION BCD SUBTRACT. SUBTRACTS TWO MUL- *
                ;* PLE-PRECISION BCD OPERANDS, ANY LENGTH.             *
                ;*     INPUT: HL=> PARAMETER BLOCK                     *
                ;*            PARAM+0,+1=ADDRESS OF OPERAND 1          *
                ;*            PARAM+2,+3=ADDRESS OF OPERAND 2          *
                ;*            PARAM+4=EVEN # OF BCD DIGITS, 0-254      *
                ;*     OUTPUT:OPERAND 1 LOCATION HOLDS RESULT          *
                ;*****************************************************
                ;
7F00 F5         BCSUBT    PUSH    AF              ;SAVE REGISTERS
7F01 C5                   PUSH    BC
7F02 D5                   PUSH    DE
7F03 E5                   PUSH    HL
7F04 DDE5                 PUSH    IX
```

```
7F06 CD7F0A    00260          CALL    0A7FH        ;***GET PB LOC'N***
7F09 E5        00270          PUSH    HL           ;TRANSFER TO IX
7F0A DDE1      00280          POP     IX
7F0C DD5E00    00290          LD      E,(IX+0)     ;GET OP 1 LOC'N
7F0F DD5601    00300          LD      D,(IX+1)
7F12 DD6E02    00310          LD      L,(IX+2)     ;GET OP 2 LOC'N
7F15 DD6603    00320          LD      H,(IX+3)
7F18 DD4E04    00330          LD      C,(IX+4)     ;GET # OF BYTES
7F1B CB39      00340          SRL     C            ;N/2
7F1D 0600      00350          LD      B,0          ;NOW IN BC
7F1F 0B        00360          DEC     BC           ;#-1
7F20 09        00370          ADD     HL,BC        ;POINT TO LAST OP2
7F21 EB        00380          EX      DE,HL        ;SWAP DE AND HL
7F22 09        00390          ADD     HL,BC        ;POINT TO LAST OP1
7F23 EB        00400          EX      DE,HL        ;SWAP BACK
7F24 41        00410          LD      B,C          ;#-1 BACK TO B
7F25 04        00420          INC     B            ;ORIGINAL NUMBER
7F26 B7        00430          OR      A            ;CLEAR CARRY FOR FIRST ADD
7F27 1A        00440 BCS010   LD      A,(DE)         ;GET OPERAND 1 BYTE
7F28 9E        00450          SBC     A,(HL)         ;SUB OPERAND 2
7F29 27        00460          DAA                    ;DECIMAL ADJUST
7F2A 12        00470          LD      (DE),A         ;STORE RESULT
7F2B 2B        00480          DEC     HL             ;POINT TO NEXT OP2
7F2C 1B        00490          DEC     DE             ;POINT TO NEXT OP1
7F2D 10F8      00500          DJNZ    BCS010         ;LOOP FOR N BYTES
7F2F DDE1      00510          POP     IX           ;RESTORE REGISTERS
7F31 E1        00520          POP     HL
7F32 D1        00530          POP     DE
7F33 C1        00540          POP     BC
7F34 F1        00550          POP     AF
7F35 C9        00560          RET                  ;RETURN TO CALLING PROG
0000           00570          END
00000 TOTAL ERRORS
```

BCSUBT DECIMAL VALUES

```
245, 197, 213, 229, 221, 229, 205, 127, 10, 229,
221, 225, 221, 94, 0, 221, 86, 1, 221, 110,
2, 221, 102, 3, 221, 78, 4, 203, 57, 6,
0, 11, 9, 235, 9, 235, 65, 4, 183, 26,
158, 39, 18, 43, 27, 16, 248, 221, 225, 225,
209, 193, 241, 201
```

CHKSUM= 131

BXBINY: BINARY TO ASCII BINARY CONVERSION

System Configuration

Model, I, Model III, Model II Stand Alone.

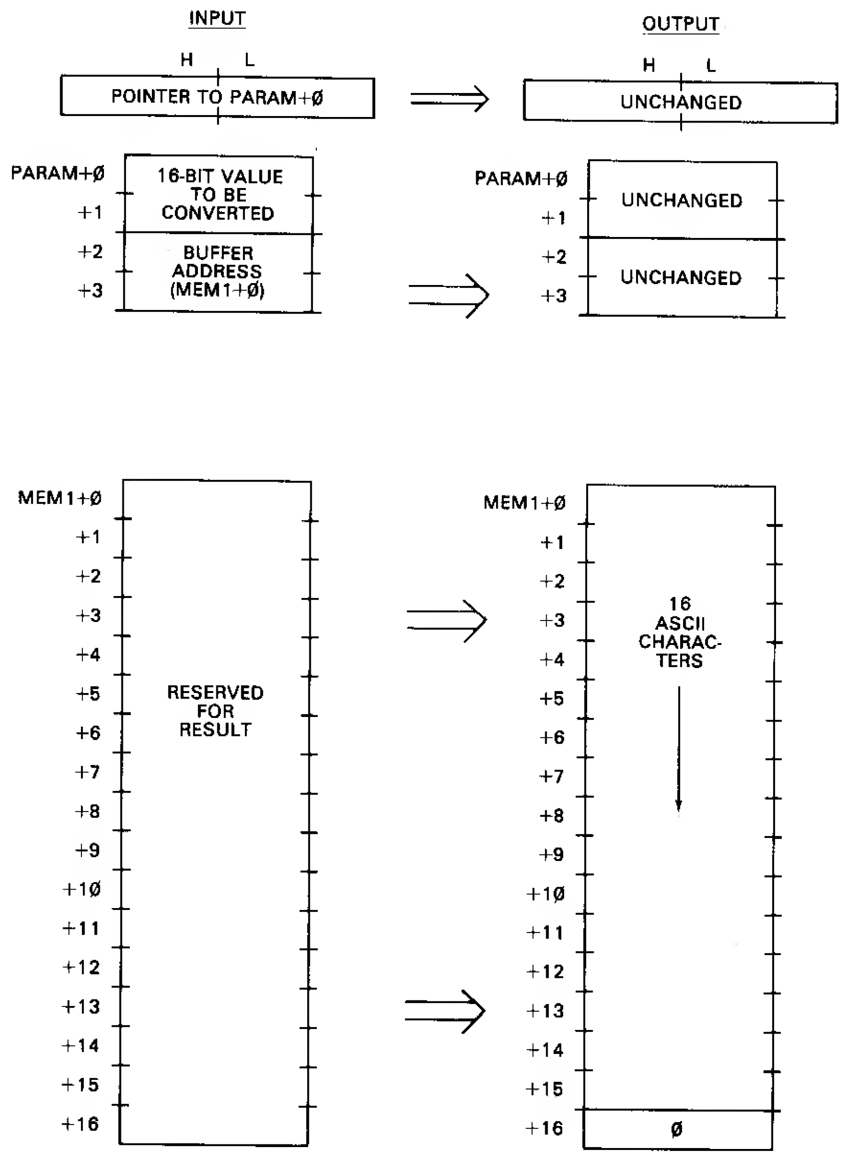
Description

BXBINY converts a 16-bit binary number to a string of ASCII binary digits. Each character in the string will be either an ASCII one (30H) or an ASCII zero (31H). The result string will be 16 bytes long, and is terminated with a byte of all zeroes. The user must specify a buffer area of 17 bytes to hold the result string.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block for BXBINY. The first two bytes of the parameter block contain the 16-bit binary value to be converted, in standard Z-80 16-bit representation, least significant byte followed by most significant byte. The next two bytes of the parameter block contain the buffer address for the 17-byte buffer that will hold the result.

On output, the buffer has been filled with the resulting string of ASCII ones and zeroes, terminated by a null. The parameter block contents remain unchanged.



Algorithm

BXBINY goes through 16 iterations to convert each of the bits in the input value to an ASCII 30H or 31H (zero or one). The value to be converted is put into register pair HL from the parameter block. For each iteration, HL is shifted left one bit position. The carry is set if the bit shifted out is a one, or reset if the bit shifted out is a zero.

The carry is tested and either a 30H (0) or 31H (1) is stored in the next buffer position. A pointer to the buffer is picked up from the parameter block and maintained in the DE register pair; it is incremented by one as each result byte is stored. The buffer is filled from low-order memory address to high-order memory address, corresponding to the processing of the bits from HL.

If the binary value to be converted was 0000000000001101, the buffer would contain 30H, 30H, 30H, 30H, 30H, 30H, 30H, 30H, 30H, 30H, 30H, 30H, 31H, 31H, 30H, 31H, 00H on return.

Sample Calling Sequence

```
NAME OF SUBROUTINE? BXBINY
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0  2  43680   VALUE TO BE CONVERTED = 1010101010100000
+ 2  2  50000
+ 4  0  0
MEMORY BLOCK 1 LOCATION? 50000
MEMORY BLOCK 1 VALUES?
+ 0  2  0    ┐
+ 2  2  0    │
+ 4  2  0    │
+ 6  2  0    │
+ 8  2  0    ├ INITIALIZE BUFFER FOR EXAMPLE
+ 10  2  0   │
+ 12  2  0   │
+ 14  2  0   │
+ 16  1  255 ┘
+ 17  0  0
MEMORY BLOCK 2 LOCATION?
MOVE SUBROUTINE TO? 37000
SUBROUTINE EXECUTED AT  37000
INPUT:              OUTPUT:
HL= 40000           HL= 40000
PARAM+ 0   160      PARAM+ 0   160 ┐
PARAM+ 1   170      PARAM+ 1   170 ├ UNCHANGED
PARAM+ 2   80       PARAM+ 2   80  │
PARAM+ 3   195      PARAM+ 3   195 ┘
MEMB1+ 0   0        MEMB1+ 0   49  ┐
MEMB1+ 1   0        MEMB1+ 1   48  │
MEMB1+ 2   0        MEMB1+ 2   49  │
MEMB1+ 3   0        MEMB1+ 3   48  │
MEMB1+ 4   0        MEMB1+ 4   49  │
MEMB1+ 5   0        MEMB1+ 5   48  │
MEMB1+ 6   0        MEMB1+ 6   49  │
MEMB1+ 7   0        MEMB1+ 7   48  │
MEMB1+ 8   0        MEMB1+ 8   49  ├ RESULT OF 1010101010100000 IN ASCII
MEMB1+ 9   0        MEMB1+ 9   48  │
MEMB1+ 10  0        MEMB1+ 10  49  │
MEMB1+ 11  0        MEMB1+ 11  48  │
MEMB1+ 12  0        MEMB1+ 12  48  │
MEMB1+ 13  0        MEMB1+ 13  48  │
MEMB1+ 14  0        MEMB1+ 14  48  │
MEMB1+ 15  0        MEMB1+ 15  48  ┘
MEMB1+ 16  255      MEMB1+ 16  0   TERMINATOR

NAME OF SUBROUTINE?
```

Notes

1. Leading ASCII zeroes may be present in the result.
2. No invalid result may occur.

Program Listing

```
7F00                        ORG     7F00H           ;0522
                    ;*****************************************************
                    ;* BINARY TO ASCII BINARY CONVERSION. CONVERTS A 16-BIT *
                    ;* BINARY VALUE TO A STRING OF ASCII ONES AND ZEROES   *
                    ;* TERMINATED BY A NULL.                                *
                    ;*     INPUT: HL=> PARAMETER BLOCK                      *
                    ;*            PARAM+0,+1=16-BIT VALUE                   *
                    ;*            PARAM+2,+3=BUFFER ADDRESS                 *
                    ;*    OUTPUT:BUFFER FILLED WITH 16 ASCII ONES AND ZER-  *
                    ;*            OES, TERMINATED BY NULL                   *
                    ;*****************************************************
                    ;
7F00 F5             BXBINY  PUSH    AF              ;SAVE REGISTERS
7F01 C5                     PUSH    BC
7F02 D5                     PUSH    DE
7F03 E5                     PUSH    HL
7F04 DDE5                   PUSH    IX
7F06 CD7F0A                 CALL    0A7FH           ;***GET PB LOC'N***
7F09 E5                     PUSH    HL              ;TRANSFER TO IX
7F0A DDE1                   POP     IX
7F0C DD6E00                 LD      L,(IX+0)        ;PUT VALUE INTO HL
7F0F DD6601                 LD      H,(IX+1)
7F12 DD5E02                 LD      E,(IX+2)        ;PUT BUFFER ADD IN DE
7F15 DD5603                 LD      D,(IX+3)
7F18 0610                   LD      B,16            ;16 ITERATIONS
7F1A 3E30           BXB010  LD      A,30H           ;ASCII ZERO
7F1C 29                     ADD     HL,HL           ;SHIFT VALUE LEFT 1 BIT
7F1D 3001                   JR      NC,BXB020       ;GO IF ZERO BIT
7F1F 3C                     INC     A               ;ASCII ONE NOW IN A
7F20 12             BXB020  LD      (DE),A          ;STORE ONE OR ZERO
7F21 13                     INC     DE              ;POINT TO NEXT SLOT
7F22 10F6                   DJNZ    BXB010          ;LOOP 'TIL END
7F24 AF                     XOR     A               ;ZERO
7F25 12                     LD      (DE),A          ;STORE NULL
7F26 DDE1                   POP     IX              ;RESTORE REGISTERS
7F28 E1                     POP     HL
7F29 D1                     POP     DE
7F2A C1                     POP     BC
7F2B F1                     POP     AF
7F2C C9                     RET                     ;RETURN TO CALLING PROG
0000                        END
00000 TOTAL ERRORS
```

```
BXBINY DECIMAL VALUES

245, 197, 213, 229, 221, 229, 205, 127, 10, 229,
221, 225, 221, 110, 0, 221, 102, 1, 221, 94,
2, 221, 86, 3, 6, 16, 62, 48, 41, 48,
1, 60, 18, 19, 16, 246, 175, 18, 221, 225,
225, 209, 193, 241, 201

CHKSUM= 34
```

BXDECL: BINARY TO ASCII DECIMAL CONVERSION

System Configuration

Model I, Model III, Model II Stand Alone.

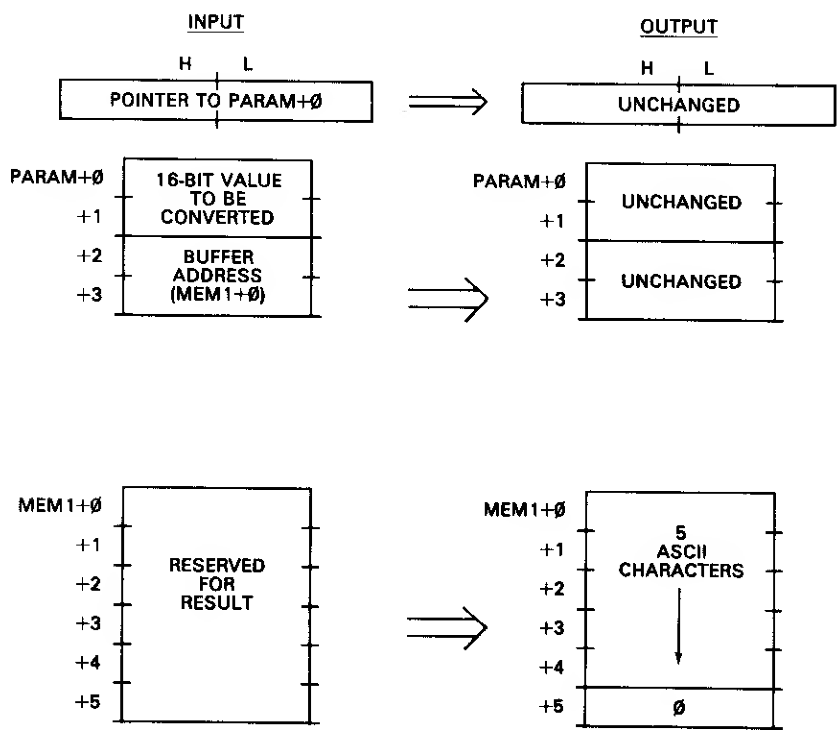
Description

BXDECL converts a 16-bit binary number to a string of ASCII decimal digits. Each character in the string will be in the range of ASCII 0 through 9 (30H through 39H). The result string will be 5 bytes long, and is terminated with a byte of all zeroes. The user must specify a buffer area of 6 bytes to hold the result string. The conversion is an "unsigned" conversion of the 16-bit value.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block for BXDECL. The first two bytes of the parameter block contain the 16-bit binary value to be converted, in standard Z-80 16-bit representation, least significant byte followed by most significant byte. The next two bytes of the parameter block contain the buffer address for the 6-byte buffer that will hold the result.

On output, the buffer has been filled with the resulting string of ASCII characters, terminated by a null. The parameter block contents remain unchanged.



Algorithm

BXDECL goes through 5 iterations to convert the input values. The value to be converted is put into register pair HL from the parameter block. For each iteration, a power of ten is subtracted from the contents of HL, starting with the largest power of ten that can be held in the 16-bit input value, 10000. Subsequent powers subtracted are 1000, 100, 10, and 1.

The first operation subtracts 10,000 as many times as possible from the original value. For each subtract, a count is incremented. If the original value were 34,567, for example, the first operation would subtract 10,000 from 34,567 four times. On the fourth time, the result would "go negative" indicating that no additional subtracts of the power could be done.

The count minus one is then added to 30H to yield the proper ASCII digit of 30H through 39H. This ASCII digit is then stored in the buffer. This operation is repeated for the five powers of ten involved.

BXDECL uses a subroutine called SUBPWR. SUBPWR is called to perform the subtracts. SUBPWR is entered with BC containing the negated power of ten to be subtracted and the current "residue" of the value to be converted in HL. A count of −1 is initially put into A. This count is incremented for each subtract. As each subtract is done, a test is made of the result. If it is negative, an add is done to restore the last result in HL. A value of 30H is then added to the value of A and the result is stored in the buffer. The pointer to the buffer is then incremented by one.

SUBPWR returns to the code in BXDECL by testing the current power of ten. It returns to one of five points at BXD010 through BXD050. This structure is necessary to avoid use of CALL instructions, which are not relocatable.

The buffer is filled from low-order memory address to high-order memory address, corresponding to the processing of the powers of ten.

If the binary value to be converted was 1010111111010011, the buffer would contain 34H, 35H, 30H, 31H, 31H, 00H on return.

Sample Calling Sequence

```
NAME OF SUBROUTINE? BXDECL
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0  2   12345  VALUE TO BE CONVERTED
+ 2  2   50000
+ 4  0   0
MEMORY BLOCK 1 LOCATION? 50000
MEMORY BLOCK 1 VALUES?
+ 0  2   0  ┐
+ 2  2   0  ├ INITIALIZE BUFFER FOR EXAMPLE
+ 4  1   0  │
+ 5  1   255┘
+ 6  0   0
MEMORY BLOCK 2 LOCATION?
MOVE SUBROUTINE TO? 45000
SUBROUTINE EXECUTED AT  45000
INPUT:              OUTPUT:
HL= 40000           HL= 40000
PARAM+ 0   57       PARAM+ 0   57 ┐
PARAM+ 1   48       PARAM+ 1   48 ├ RESULT OF 12345 IN ASCII
PARAM+ 2   80       PARAM+ 2   80 │
PARAM+ 3   195      PARAM+ 3   195┘
```

```
MEMB1+ 0   0       MEMB1+ 0   49 ┐
MEMB1+ 1   0       MEMB1+ 1   50 │
MEMB1+ 2   0       MEMB1+ 2   51 ├ UNCHANGED
MEMB1+ 3   0       MEMB1+ 3   52 │
MEMB1+ 4   0       MEMB1+ 4   53 ┘
MEMB1+ 5   255     MEMB1+ 5   0
```

```
NAME OF SUBROUTINE?
```

Notes

1. Leading ASCII zeroes may be present in the result.
2. No invalid result may occur.

Program Listing

```
7F00                        ORG     7F00H           ;0522
             ;*****************************************************
             ;* BINARY TO ASCII DECIMAL CONVERSION. CONVERTS A 16-BIT*
             ;* BINARY VALUE TO A STRING OF ASCII DECIMAL DIGITS TER-*
             ;* MINATED BY A NULL.                                  *
             ;*    INPUT: HL=> PARAMETER BLOCK                      *
             ;*           PARAM+0,+1=16 BIT VALUE                   *
             ;*           PARAM+2,+3=BUFFER ADDRESS                 *
             ;*   OUTPUT:BUFFER FILLED WITH 5 ASCII DIGITS, TERM-   *
             ;*           INATED BY NULL                            *
             ;*****************************************************
             ;
7F00 F5      BXDECL   PUSH    AF              ;SAVE REGISTERS
7F01 C5               PUSH    BC
7F02 D5               PUSH    DE
7F03 E5               PUSH    HL
7F04 DDE5             PUSH    IX
7F06 CD7F0A           CALL    0A7FH           ;***GET PB LOC'N***
7F09 E5               PUSH    HL              ;TRANSFER TO IX
7F0A DDE1             POP     IX
7F0C DD6E00           LD      L,(IX+0)        ;PUT VALUE INTO HL
7F0F DD6601           LD      H,(IX+1)
7F12 DD5E02           LD      E,(IX+2)        ;PUT BUFFER ADD IN DE
7F15 DD5603           LD      D,(IX+3)
7F18 01F0D8           LD      BC,-10000       ;10 TO THE FOURTH
7F1B 181D             JR      SUBPWR          ;FIND FIRST DIGIT
7F1D 0118FC  BXD010   LD      BC,-1000        ;10 TO THE THIRD
7F20 1818             JR      SUBPWR          ;FIND SECOND DIGIT
7F22 019CFF  BXD020   LD      BC,-100         ;10 TO THE SECOND
7F25 1813             JR      SUBPWR          ;FIND THIRD DIGIT
7F27 01F6FF  BXD030   LD      BC,-10          ;10 TO THE FIRST
7F2A 180E             JR      SUBPWR          ;FIND FOURTH DIGIT
7F2C 01FFFF  BXD040   LD      BC,-1           ;10 TO THE ZEROTH
7F2F 1809             JR      SUBPWR          ;FIND LAST DIGIT
7F31 AF      BXD050   XOR     A               ;ZERO
7F32 12               LD      (DE),A          ;STORE NULL
7F33 DDE1             POP     IX              ;RESTORE REGISTERS
7F35 E1               POP     HL
7F36 D1               POP     DE
7F37 C1               POP     BC
7F38 F1               POP     AF
7F39 C9               RET                     ;RETURN TO CALLING PROG
7F3A 3EFF    SUBPWR   LD      A,0FFH          ;-1 TO A
7F3C 3C      SUB010   INC     A               ;BUMP DIGIT COUNT
7F3D 09               ADD     HL,BC           ;SUBTRACT PWR OF TEN
7F3E 38FC             JR      C,SUB010        ;GO IF NOT NEGATIVE
7F40 B7               OR      A               ;CLEAR CARRY
```

```
7F41 ED42       00570           SBC     HL,BC           ;RESTORE LAST RESULT
7F43 C630       00580           ADD     A,30H           ;CONVERT TO ASCII
7F45 12         00590           LD      (DE),A          ;STORE IN BUFFER
7F46 13         00600           INC     DE              ;POINT TO NEXT SLOT
7F47 79         00610           LD      A,C             ;GET LSB OF PWR
7F48 FEF0       00620           CP      0F0H            ;TEST FOR -10000
7F4A 28D1       00630           JR      Z,BXD010        ;GO IF -10000
7F4C FE18       00640           CP      18H             ;TEST FOR -1000
7F4E 28D2       00650           JR      Z,BXD020        ;GO IF -1000
7F50 FE9C       00660           CP      9CH             ;TEST FOR -100
7F52 28D3       00670           JR      Z,BXD030        ;GO IF -100
7F54 FEF6       00680           CP      0F6H            ;TEST FOR -10
7F56 28D4       00690           JR      Z,BXD040        ;GO IF -10
7F58 18D7       00700           JR      BXD050          ;MUST BE -1
0000            00710           END
00000 TOTAL ERRORS
```

```
BXDECL DECIMAL VALUES

245, 197, 213, 229, 221, 229, 205, 127, 10, 229,
221, 225, 221, 110, 0, 221, 102, 1, 221, 94,
2, 221, 86, 3, 1, 240, 216, 24, 29, 1,
24, 252, 24, 24, 1, 156, 255, 24, 19, 1,
246, 255, 24, 14, 1, 255, 255, 24, 9, 175,
18, 221, 225, 225, 209, 193, 241, 201, 62, 255,
60, 9, 56, 252, 183, 237, 66, 198, 48, 18,
19, 121, 254, 240, 40, 209, 254, 24, 40, 210,
254, 156, 40, 211, 254, 246, 40, 212, 24, 215

CHKSUM= 190
```

BXHEXD: BINARY TO ASCII HEXADECIMAL CONVERSION

System Configuration

Model I, Model III, Model II Stand Alone.

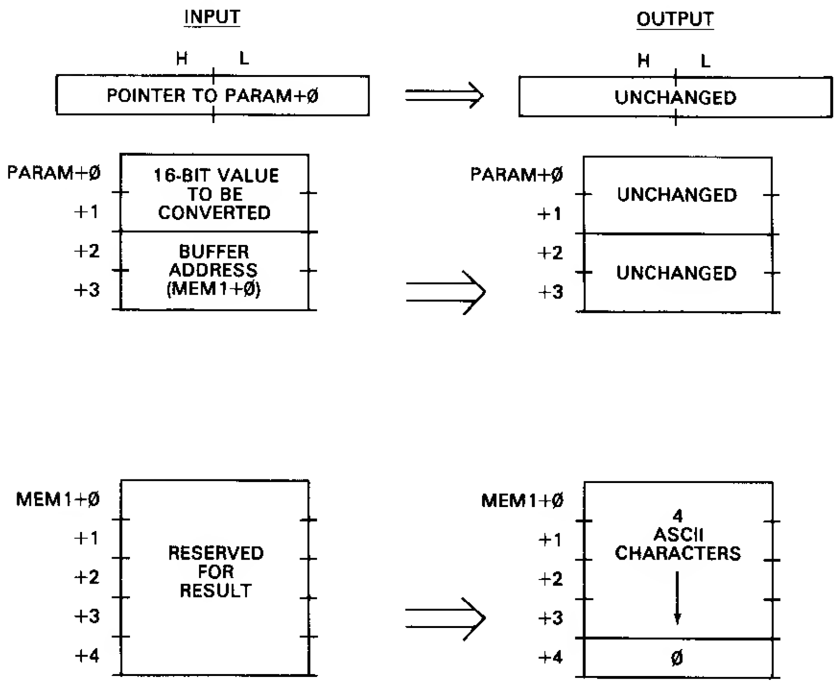
Description

BXHEXD converts a 16-bit binary number to a string of ASCII hexadecimal digits. Each character in the string will be in the range of ASCII 0 through 9 (30H through 37H) or ASCII A through F (41H through 46H). The result string will be 4 bytes long, and is terminated with a byte of all zeroes. The user must specify a buffer area of 5 bytes to hold the result string.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block for BXHEXD. The first two bytes of the parameter block contain the 16-bit binary value to be converted, in standard Z-80 16-bit representation, least significant byte followed by most significant byte. The next two bytes of the parameter block contain the buffer address for the 5-byte buffer that will hold the result.

On output, the buffer has been filled with the resulting string of ASCII characters, terminated by a null. The parameter block contents remain unchanged.



Algorithm

BXHEXD goes through 4 iterations to convert each of the bits in the input value to an ASCII 30H through 39H (zero through nine) or 41H through 46H (A through F). The value to be converted is put into register pair HL from the parameter block. For each iteration, HL is shifted four bit positions with the four bits from the shift going into the four least significant bits of the A register.

A test is then made of the value in A. If it is in the range 0 through 9, a "bias" value of 30H is set aside. If it is in the range of 10 through 15, a bias value of 37H is saved. The bias value is then added to the contents of A, converting the three bits to an ASCII octal digit of 30H through 39H or 41H through 46H. The ASCII character is then stored in the user buffer. A pointer to the buffer is picked up from the parameter block and maintained in the DE register pair; it is incremented by one as each result byte is stored. The buffer is filled from low-order memory address to high-order memory address, corresponding to the processing of the bits from HL.

If the binary value to be converted was 1111000000111101, the buffer would contain 45H, 30H, 33H, 44H, 00H on return.

Sample Calling Sequence

```
NAME OF SUBROUTINE? BXHEXD
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0  2  4660   VALUE TO BE CONVERTED
```

```
+ 2   2   50000
+ 4   0   0
MEMORY BLOCK 1 LOCATION? 50000
MEMORY BLOCK 1 VALUES?
+ 0   2   0
+ 2   2   0      INITIALIZE BUFFER FOR EXAMPLE
+ 4   1   255
+ 5   0   0
MEMORY BLOCK 2 LOCATION?
MOVE SUBROUTINE TO? 37777
SUBROUTINE EXECUTED AT  37777
INPUT:                 OUTPUT:
HL= 40000              HL= 40000
PARAM+ 0   52          PARAM+ 0   52
PARAM+ 1   18          PARAM+ 1   18    UNCHANGED
PARAM+ 2   80          PARAM+ 2   80
PARAM+ 3   195         PARAM+ 3   195
MEMB1+ 0   0           MEMB1+ 0   49
MEMB1+ 1   0           MEMB1+ 1   50
MEMB1+ 2   0           MEMB1+ 2   51    RESULT OF 1234 IN ASCII
MEMB1+ 3   0           MEMB1+ 3   52
MEMB1+ 4   255         MEMB1+ 4   0     TERMINATOR

NAME OF SUBROUTINE?
```

Notes

1. Leading ASCII zeroes may be present in the result.
2. No invalid result may occur.

Program Listing

```
7F00                          ORG     7F00H             ;0522
                  ;*******************************************************
                  ;* BINARY TO ASCII HEXADECIMAL CONVERSION. CONVERTS A   *
                  ;* 16-BIT BINARY VALUE TO A STRING OF ASCII HEX DIGITS  *
                  ;* TERMINATED BY A NULL.                                *
                  ;*     INPUT: HL=> PARAMETER BLOCK                      *
                  ;*            PARAM+0,+1=16-BIT VALUE                   *
                  ;*            PARAM+2,+3=BUFFER ADDRESS                 *
                  ;*    OUTPUT:BUFFER FILLED WITH FOUR ASCII HEX DIGITS,  *
                  ;*            TERMINATED BY NULL                        *
                  ;*******************************************************
                  ;
7F00 F5           BXHEXD  PUSH    AF                ;SAVE REGISTERS
7F01 C5                   PUSH    BC
7F02 D5                   PUSH    DE
7F03 E5                   PUSH    HL
7F04 DDE5                 PUSH    IX
7F06 CD7F0A               CALL    0A7FH             ;***GET PB LOC'N***
7F09 E5                   PUSH    HL
7F0A DDE1                 POP     IX
7F0C DD6E00               LD      L,(IX+0)          ;PUT VALUE INTO HL
7F0F DD6601               LD      H,(IX+1)
7F12 DD5E02               LD      E,(IX+2)          ;PUT BUFFER ADD IN DE
7F15 DD5603               LD      D,(IX+3)
7F18 0604                 LD      B,4               ;ITERATION COUNT
7F1A AF           BXH010  XOR     A                   ;ZERO A
7F1B 29                   ADD     HL,HL               ;SHIFT OUT BIT LEFT
7F1C 17                   RLA                         ;SHIFT INTO A
7F1D 29                   ADD     HL,HL
7F1E 17                   RLA
7F1F 29                   ADD     HL,HL
```

```
7F20 17        00410         RLA
7F21 29        00420         ADD     HL,HL
7F22 17        00430         RLA
7F23 F5        00440         PUSH    AF            ;SAVE 4 BITS
7F24 0E30      00450         LD      C,30H         ;ASCII ZERO
7F26 D60A      00460         SUB     10            ;TEST FOR 0 - 9
7F28 CB7F      00470         BIT     7,A           ;TEST SIGN
7F2A 2002      00480         JR      NZ,BXH020     ;GO IF 0-9
7F2C 0E37      00490         LD      C,37H         ;ADJUSTMENT FOR A - F
7F2E F1        00500 BXH020  POP     AF            ;RESTORE ORIGINAL BITS
7F2F 81        00510         ADD     A,C           ;ADD IN ASCII BIAS
7F30 12        00520         LD      (DE),A        ;STORE CHARACTER
7F31 13        00530         INC     DE            ;POINT TO NEXT SLOT
7F32 10E6      00540         DJNZ    BXH010        ;LOOP 'TIL 4
7F34 AF        00550         XOR     A             ;ZERO
7F35 12        00560         LD      (DE),A        ;STORE NULL
7F36 DDE1      00570         POP     IX            ;RESTORE REGISTERS
7F38 E1        00580         POP     HL
7F39 D1        00590         POP     DE
7F3A C1        00600         POP     BC
7F3B F1        00610         POP     AF
7F3C C9        00620         RET                   ;RETURN TO CALLING PROG
0000           00630         END
00000 TOTAL ERRORS

                    BXHEXD DECIMAL VALUES

245, 197, 213, 229, 221, 229, 205, 127, 10, 229,
221, 225, 221, 110, 0, 221, 102, 1, 221, 94,
2, 221, 86, 3, 6, 4, 175, 41, 23, 41,
23, 41, 23, 41, 23, 245, 14, 48, 214, 10,
203, 127, 32, 2, 14, 55, 241, 129, 18, 19,
16, 230, 175, 18, 221, 225, 225, 209, 193, 241,
201

CHKSUM= 231
```

BXOCTL: BINARY TO ASCII OCTAL CONVERSION

System Configuration

Model I, Model III, Model II Stand Alone.

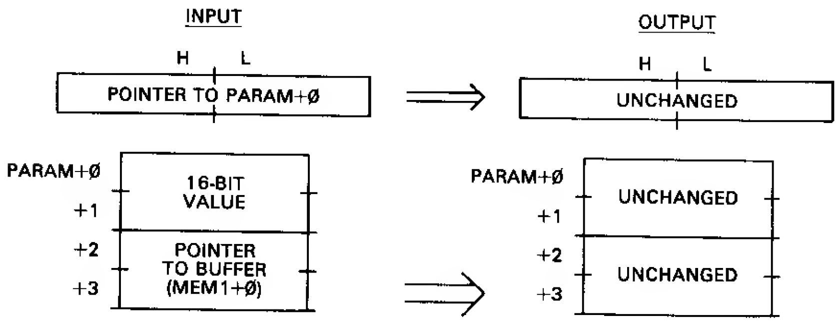
Description

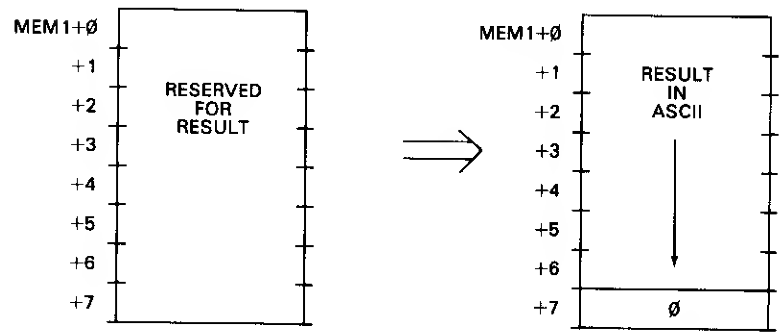
BXOCTL converts a 16-bit binary number to a string of ASCII octal digits. Each character in the string will be in the range of ASCII 0 through 7 (30H through 37H). The result string will be 6 bytes long, and is terminated with a byte of all zeroes. The user must specify a buffer area of 7 bytes to hold the result string.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block for BXOCTL. The first two bytes of the parameter block contain the 16-bit binary value to be converted, in standard Z-80 16-bit representation, least significant byte followed by most significant byte. The next two bytes of the parameter block contain the buffer address for the 7-byte buffer that will hold the result.

On output, the buffer has been filled with the resulting string of ASCII characters, terminated by a null. The parameter block contents remain unchanged.





Algorithm

BXOCTL goes through 6 iterations to convert each of the bits in the input value to an ASCII 30H through 37H (zero through seven). The value to be converted is put into register pair HL from the parameter block. For each iteration except the first, HL is shifted three bit positions with the three bits from the shift going into the three least significant bits of the A register. (The first iteration performs only one shift to handle the leading octal digit of 0 or 1.)

A value of 30H is then added to the contents of A. This converts the three bits to an ASCII octal digit of 30H through 37H. The ASCII character is then stored in the user buffer. A pointer to the buffer is picked up from the parameter block and maintained in the DE register pair; it is incremented by one as each result byte is stored. The buffer is filled from low-order memory address to high-order memory address, corresponding to the processing of the bits from HL.

If the binary value to be converted was 1000000000001101, the buffer would contain 31H, 30H, 30H, 30H, 31H, 35H, 00H on return.

Sample Calling Sequence

```
NAME OF SUBROUTINE? BXOCTL
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
```

```
PARAMETER BLOCK VALUES?
+ 0   2   12345  VALUE TO BE CONVERTED = 030071 OCTAL
+ 2   2   45000
+ 4   0   0
MEMORY BLOCK 1 LOCATION? 45000
MEMORY BLOCK 1 VALUES?
+ 0   1   255
+ 1   1   255
+ 2   1   255
+ 3   1   255   INITIALIZE BUFFER FOR EXAMPLE
+ 4   1   255
+ 5   1   255
+ 6   1   255
+ 7   0   0
MEMORY BLOCK 2 LOCATION?
MOVE SUBROUTINE TO? 37777
SUBROUTINE EXECUTED AT  37777
INPUT:                 OUTPUT:
HL= 40000              HL= 40000
PARAM+ 0   57          PARAM+ 0   57
PARAM+ 1   48          PARAM+ 1   48
PARAM+ 2   200         PARAM+ 2   200
PARAM+ 3   175         PARAM+ 3   175
MEMB1+ 0   255         MEMB1+ 0   48
MEMB1+ 1   255         MEMB1+ 1   51
MEMB1+ 2   255         MEMB1+ 2   48   RESULT = 030071 IN ASCII
MEMB1+ 3   255         MEMB1+ 3   48
MEMB1+ 4   255         MEMB1+ 4   55
MEMB1+ 5   255         MEMB1+ 5   49
MEMB1+ 6   255         MEMB1+ 6   0   TERMINATOR

NAME OF SUBROUTINE?
```

Notes

1. Leading ASCII zeroes may be present in the result.
2. No invalid result may occur.
3. The most significant ASCII character will always be either a zero (30H) or a one (31H) since 16 bits is not an integer multiple of 3 bits.

Program Listing

```
7F00                          ORG     7F00H           ;0522
                ;*******************************************************
                ;* BINARY TO ASCII OCTAL CONVERSION. CONVERTS A 16-BIT  *
                ;* BINARY VALUE TO A STRING OF ASCII OCTAL DIGITS TERM- *
                ;* INATED BY A NULL.                                    *
                ;*     INPUT: HL=> PARAMETER BLOCK                      *
                ;*            PARAM+0,+1=16-BIT VALUE                   *
                ;*            PARAM+2,+3=BUFFER ADDRESS                 *
                ;*     OUTPUT:BUFFER FILLED WITH SIX ASCII OCTAL DIG-   *
                ;*            ITS TERMINATED BY NULL                    *
                ;*******************************************************
                ;
7F00 F5         BXOCTL        PUSH    AF              ;SAVE REGISTERS
7F01 C5                       PUSH    BC
7F02 D5                       PUSH    DE
7F03 E5                       PUSH    HL
7F04 DDE5                     PUSH    IX
7F06 CD7F0A                   CALL    0A7FH           ;***GET PB LOC'N***
7F09 E5                       PUSH    HL
7F0A DDE1                     POP     IX
```

```
7F0C DD6E00     00300           LD      L,(IX+0)        ;PUT VALUE INTO HL
7F0F DD6601     00310           LD      H,(IX+1)
7F12 DD5E02     00320           LD      E,(IX+2)        ;PUT BUFFER ADD IN DE
7F15 DD5603     00330           LD      D,(IX+3)
7F18 0606       00340           LD      B,6             ;ITERATION COUNT
7F1A AF         00350           XOR     A               ;ZERO A
7F1B 1805       00360           JR      BX0020          ;FOR FIRST DIGIT
7F1D AF         00370 BX0010    XOR     A                 ;ZERO A
7F1E 29         00380           ADD     HL,HL             ;SHIFT OUT BIT LEFT
7F1F 17         00390           RLA                       ;SHIFT INTO A
7F20 29         00400           ADD     HL,HL
7F21 17         00410           RLA
7F22 29         00420 BX0020    ADD     HL,HL
7F23 17         00430           RLA
7F24 0E30       00440           LD      C,30H             ;ASCII ZERO
7F26 81         00450           ADD     A,C               ;ADD IN ASCII BIAS
7F27 12         00460           LD      (DE),A            ;STORE CHARACTER
7F28 13         00470           INC     DE                ;POINT TO NEXT SLOT
7F29 10F2       00480           DJNZ    BX0010            ;LOOP 'TIL 6
7F2B AF         00490           XOR     A               ;ZERO
7F2C 12         00500           LD      (DE),A          ;STORE NULL
7F2D DDE1       00510           POP     IX              ;RESTORE REGISTERS
7F2F E1         00520           POP     HL
7F30 D1         00530           POP     DE
7F31 C1         00540           POP     BC
7F32 F1         00550           POP     AF
7F33 C9         00560           RET                     ;RETURN TO CALLING PROG
0000            00570           END
00000 TOTAL ERRORS
```

```
BXOCTL DECIMAL VALUES

245, 197, 213, 229, 221, 229, 205, 127, 10, 229,
221, 225, 221, 110, 0, 221, 102, 1, 221, 94,
2, 221, 86, 3, 6, 6, 175, 24, 5, 175,
41, 23, 41, 23, 41, 23, 14, 48, 129, 18,
19, 16, 242, 175, 18, 221, 225, 225, 209, 193,
241, 201

CHKSUM= 10
```

CHKSUM: CHECKSUM MEMORY

System Configuration

Model I, Model III, Model II Stand Alone.

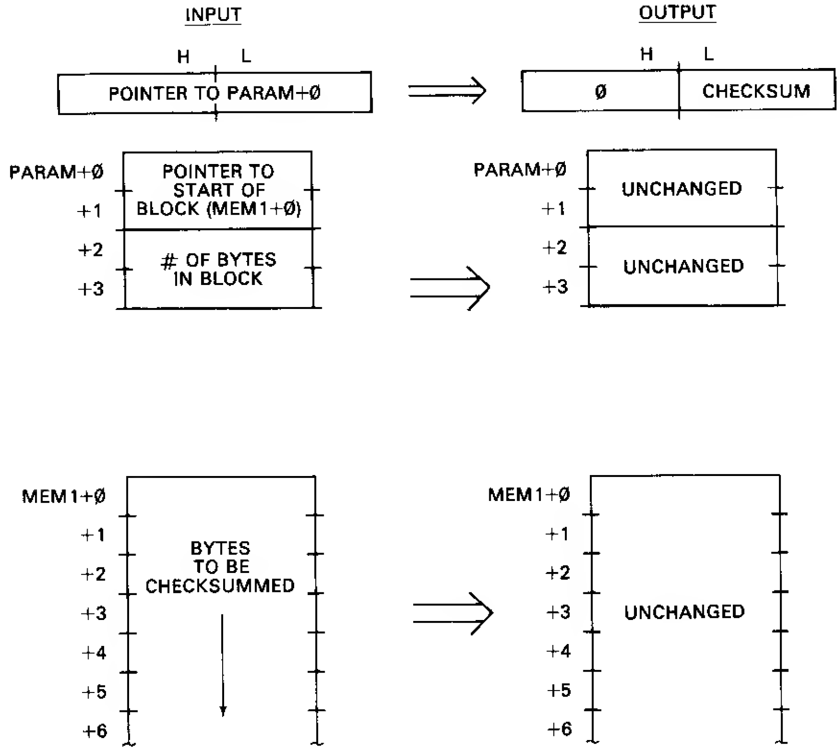
Description

CHKSUM checksums a block of memory for verification of data. The checksum performed is a simple additive 8-bit checksum.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block define the starting address for the block of memory to be checksummed in standard Z-80 address format, least significant byte followed by most significant byte. The next two bytes of the parameter block contain the number of bytes in the block to be checksummed.

On output, HL contains the checksum of the block of memory.



Algorithm

The CHKSUM subroutine first picks up the number of bytes in the block and puts it into the HL register pair. Next, the starting address is put into the IX register. The A register is cleared for the checksum.

The loop at CHK010 adds in each byte from the memoy block. The count in HL is decremented by a subtract of one in BC, and the pointer in IX is adjusted to point to the next memory byte.

Sample Calling Sequence

```
NAME OF SUBROUTINE? CHKSUM
HL VALUE? 43000
PARAMETER BLOCK LOCATION? 43000
PARAMETER BLOCK VALUES?
+ 0  2  45000   START OF BLOCK
+ 2  2  8       8 BYTES IN BLOCK
+ 4  0  0
MEMORY BLOCK 1 LOCATION? 45000
MEMORY BLOCK 1 VALUES?
+ 0  1  1   ┐
+ 1  1  2   │
+ 2  1  4   │
+ 3  1  8   │
+ 4  1  16  ├ SAMPLE DATA
+ 5  1  32  │
+ 6  1  64  │
+ 7  1  128 ┘
+ 8  0  0
```

```
MEMORY BLOCK 2 LOCATION?
MOVE SUBROUTINE TO? 46000
SUBROUTINE EXECUTED AT  46000
INPUT:                OUTPUT:
HL= 43000             HL= 255    CHECKSUM = 1 + 2 + 4 . . . + 128
PARAM+ 0   200        PARAM+ 0   200
PARAM+ 1   175        PARAM+ 1   175
PARAM+ 2   8          PARAM+ 2   8
PARAM+ 3   0          PARAM+ 3   0
MEMB1+ 0   1          MEMB1+ 0   1
MEMB1+ 1   2          MEMB1+ 1   2      UNCHANGED
MEMB1+ 2   4          MEMB1+ 2   4
MEMB1+ 3   8          MEMB1+ 3   8
MEMB1+ 4   16         MEMB1+ 4   16
MEMB1+ 5   32         MEMB1+ 5   32
MEMB1+ 6   64         MEMB1+ 6   64
MEMB1+ 7   128        MEMB1+ 7   128

NAME OF SUBROUTINE?
```

Notes

1. The CHKSUM subroutine is used to compute the checksum for all subroutines in this book.

Program Listing

```
7F00                        ORG     7F00H            ;0522
                  ;*******************************************************
                  ;* CHECKSUM MEMORY. CHECKSUMS A BLOCK OF MEMORY.       *
                  ;*     INPUT: HL=>PARAMETER BLOCK                      *
                  ;*            PARAM+0,+1=STARTING ADDRESS OF BLOCK     *
                  ;*            PARAM+2,+3=# OF BYTES IN BLOCK           *
                  ;*     OUTPUT:HL=ADDITIVE CHECKSUM                     *
                  ;*******************************************************
                  ;
7F00 F5           CHKSUM    PUSH    AF               ;SAVE REGISTERS
7F01 C5                     PUSH    BC
7F02 D5                     PUSH    DE
7F03 DDE5                   PUSH    IX
7F05 CD7F0A                 CALL    0A7FH            ;***GET PB LOC'N***
7F08 E5                     PUSH    HL               ;TRANSFER HL TO IX
7F09 DDE1                   POP     IX
7F0B DD6E02                 LD      L,(IX+2)         ;GET # OF BYTES
7F0E DD6603                 LD      H,(IX+3)
7F11 DD5E00                 LD      E,(IX+0)         ;GET STARTING ADDRESS
7F14 DD5601                 LD      D,(IX+1)
7F17 D5                     PUSH    DE               ;TRANSFER TO IX
7F18 DDE1                   POP     IX
7F1A 010100                 LD      BC,1             ;DECREMENT VALUE
7F1D AF                     XOR     A                ;CLEAR CHECKSUM
7F1E DD8600       CHK010    ADD     A,(IX+0)           ;CHECKSUM
7F21 DD23                   INC     IX                 ;BUMP ADDRESS PNTR
7F23 B7                     OR      A                  ;CLEAR CARRY
7F24 ED42                   SBC     HL,BC              ;DECREMENT COUNT
7F26 20F6                   JR      NZ,CHK010          ;GO IF NOT DONE
7F28 6F                     LD      L,A              ;MOVE CHECKSUM TO HL
7F29 2600                   LD      H,0
7F2B DDE1                   POP     IX               ;RESTORE REGISTERS
7F2D D1                     POP     DE
7F2E C1                     POP     BC
7F2F F1                     POP     AF
```

```
7F30 C39A0A    00450          JP       0A9AH                ;***RETURN STATUS***
7F33 C9        00460          RET                           ;NON-BASIC RETURN
0000           00470          END
00000 TOTAL ERRORS
```

```
CHKSUM DECIMAL VALUES


 245, 197, 213, 221, 229, 205, 127, 10, 229, 221,
 225, 221, 110, 2, 221, 102, 3, 221, 94, 0,
 221, 86, 1, 213, 221, 225, 1, 1, 0, 175,
 221, 134, 0, 221, 35, 183, 237, 66, 32, 246,
 111, 38, 0, 221, 225, 209, 193, 241, 195, 154,
 10, 201

CHKSUM= 245
```

CLEARS: CLEAR SCREEN

System Configuration

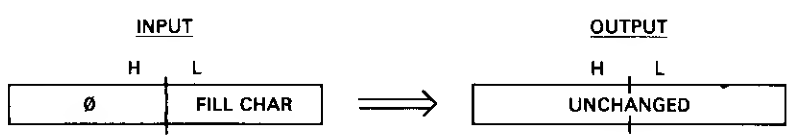
Model I, Model III.

Description

CLEARS clears the video screen or outputs a given character to fill the screen. For a clear screen, the character is normally a blank (20H), or a graphics "all off" character (080H).

Input/Output Parameters

On input, the HL register pair contains the character to be used in the fill. (The L register contains the 8-bit character while the H register contains zero.) On output, the screen has been cleared or filled.



Algorithm

The CLEARS subroutine is similar to a "fill memory" subroutine except that the memory to fill is defined as 3C00H through 3FFFH.

The start of video display memory, 3C00H, is put into HL and the character for the fill is transferred to B. The loop at CLE010 fills a byte at a time. For each fill, the video display memory pointer is incremented by one and the contents of the H register are tested. If H holds 40H, the last screen location has been filled.

Sample Calling Sequence

```
NAME OF SUBROUTINE? CLEARS
HL VALUE? 65  CLEAR CHARACTER OF "A"
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 37000
SUBROUTINE EXECUTED AT  37000
INPUT:              OUTPUT:
HL= 65              HL= 65  UNCHANGED

NAME OF SUBROUTINE?
```

Notes

1. The CLEARS subroutine clears the screen in approximately 21 milliseconds.

Program Listing

```
7F00          00100          ORG     7F00H            ;0520
              00110 ;*******************************************************
              00120 ;* CLEAR SCREEN. CLEARS THE SCREEN OR FILLS THE SCREEN  *
              00130 ;* WITH ANY GIVEN CHARACTER.                            *
              00140 ;*     INPUT: HL=CHARACTER FOR CLEAR,NORMALLY 20H OR 80H *
              00150 ;*     OUTPUT:NONE                                      *
              00160 ;*******************************************************
              00170 ;
7F00 F5       00180 CLEARS   PUSH    AF               ;SAVE REGISTERS
7F01 C5       00190          PUSH    BC
7F02 E5       00200          PUSH    HL
7F03 CD7F0A   00210          CALL    0A7FH            ;***GET CLEAR CHAR***
7F06 45       00220          LD      B,L              ;TRANSFER TO B
7F07 21003C   00230          LD      HL,3C00H         ;START OF SCREEN ADDRESS
7F0A 70       00240 CLE010   LD      (HL),B             ;FILL SCREEN BYTE
7F0B 23       00250          INC     HL                 ;BUMP SCREEN POINTER
7F0C 7C       00260          LD      A,H                ;GET MS BYTE OF POINTER
7F0D FE40     00270          CP      40H                ;TEST FOR END+1
7F0F 20F9     00280          JR      NZ,CLE010          ;CONTINUE IF NOT END
7F11 E1       00290          POP     HL               ;RESTORE REGISTERS
7F12 C1       00300          POP     BC
7F13 F1       00310          POP     AF
7F14 C9       00320          RET                      ;RETURN TO CALLING PROGRAM
0000          00330          END
00000 TOTAL ERRORS
```

```
CLEARS DECIMAL VALUES


245, 197, 229, 205, 127, 10, 69, 33, 0, 60,
112, 35, 124, 254, 64, 32, 249, 225, 193, 241,
201

CHKSUM= 89
```

CSCLNE: CLEAR SCREEN LINES

System Configuration

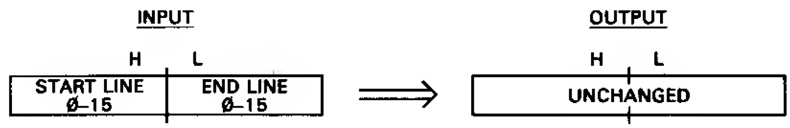
Model I, Model III.

Description

CSCLNE clears from one to 16 screen lines with blank (20H) characters. The lines cleared may be any set of contiguous lines on the screen, starting with any given line.

Input/Output Parameters

On input, the H register contains the start line number, from 0 through 15, and the L register contains the end line number, from 0 through 15. On output, the designated screen lines have been cleared and HL is unchanged.



Algorithm

The CSCLNE subroutine first finds the total number of lines involved in the clear. The start line number is subtracted from the end line number, and this value is incremented by one. Next, this line count is multiplied by 64 to find the total number of video display memory bytes to be cleared (CSC010).

The starting video memory location is then found by multiplying the starting line number by 64 (CSC020) and adding this value to the screen start location of 3C00H.

The loop at CSC030 stores a blank character in the screen locations involved. HL contains the pointer to screen memory, which is incremented each time through the loop, and DE contains the number of screen bytes to be filled. The count in DE is tested for zero by the "load and OR" operation.

Sample Calling Sequence

```
NAME OF SUBROUTINE? CSCLNE
HL VALUE? 1800  START LINE = 7, END LINE = 8
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 55000
SUBROUTINE EXECUTED AT  55000
INPUT:             OUTPUT:
HL= 1800           HL= 1800  UNCHANGED

NAME OF SUBROUTINE?
```

Notes

1. Use the CLEARS subroutine to clear the entire screen.
2. No check is made on the validity of the line numbers in HL. If the wrong values are used, the system may crash.
3. The end line number must be greater or equal to the start line number.
4. Use an 80H in location 7F23H for a "graphics" clear.

```
7F00                        ORG     7F00H             ;0522
                  ;*****************************************************
                  ;* CLEAR SCREEN LINE. CLEARS THE SCREEN FROM A GIVEN  *
                  ;* START LINE THROUGH A GIVEN END LINE.                *
                  ;*     INPUT: HL=START LINE(H), END LINE(L) 0-15       *
                  ;*     OUTPUT:SCREEN LINES CLEARED WITH BLANKS         *
                  ;*****************************************************
                  ;
7F00 F5           CSCLNE    PUSH    AF                ;SAVE REGISTERS
7F01 C5                     PUSH    BC
7F02 D5                     PUSH    DE
7F03 E5                     PUSH    HL
7F04 CD7F0A                 CALL    0A7FH             ;***GET LINE NOS***
7F07 E5                     PUSH    HL                ;SAVE
7F08 7D                     LD      A,L               ;END LINE NUMBER
7F09 94                     SUB     H                 ;END-START
7F0A 3C                     INC     A                 ;TOTAL NUMBER OF LINES
7F0B 6F                     LD      L,A               ;TOTAL TO L
7F0C 2600                   LD      H,0               ;NOW IN HL
7F0E 0606                   LD      B,6               ;ITERATION COUNT
7F10 29           CSC010    ADD     HL,HL               ;# LINES * 64=# CHARS
7F11 10FD                   DJNZ    CSC010              ;LOOP 'TIL DONE
7F13 E5                     PUSH    HL                ;TRANSFER # CHARACTERS
7F14 D1                     POP     DE                ;NOW IN DE
7F15 E1                     POP     HL                ;ORIGINAL LINE #S
7F16 6C                     LD      L,H               ;START LINE #
7F17 2600                   LD      H,0               ;NOW IN HL
7F19 0606                   LD      B,6               ;ITERATION COUNT
7F1B 29           CSC020    ADD     HL,HL               ;FIND DISPLACEMENT
7F1C 10FD                   DJNZ    CSC020              ;LOOP 'TIL DONE
7F1E 01003C                 LD      BC,3C00H          ;START OF SCREEN
7F21 09                     ADD     HL,BC             ;FIND START MEMORY LOC'N
7F22 3620         CSC030    LD      (HL),' '            ;STORE BLANK
7F24 23                     INC     HL                  ;BUMP SCREEN POINTER
7F25 1B                     DEC     DE                  ;DECREMENT COUNT
7F26 7A                     LD      A,D                 ;TEST COUNT
7F27 B3                     OR      E
7F28 20F8                   JR      NZ,CSC030           ;GO IF DE NE ZERO
7F2A E1                     POP     HL                ;RESTORE REGISTERS
7F2B D1                     POP     DE
7F2C C1                     POP     BC
7F2D F1                     POP     AF
7F2E C9                     RET                       ;RETURN TO CALLING PROG
0000                        END
00000 TOTAL ERRORS
```

CSCLNE DECIMAL VALUES

```
245, 197, 213, 229, 205, 127, 10, 229, 125, 148,
60, 111, 38, 0, 6, 6, 41, 16, 253, 229,
209, 225, 108, 38, 0, 6, 6, 41, 16, 253,
1, 0, 60, 9, 54, 32, 35, 27, 122, 179,
32, 248, 225, 209, 193, 241, 201
```

CHKSUM= 138

CSTRNG: STRING COMPARE

System Configuration

Model I, Model III, Model II Stand Alone.

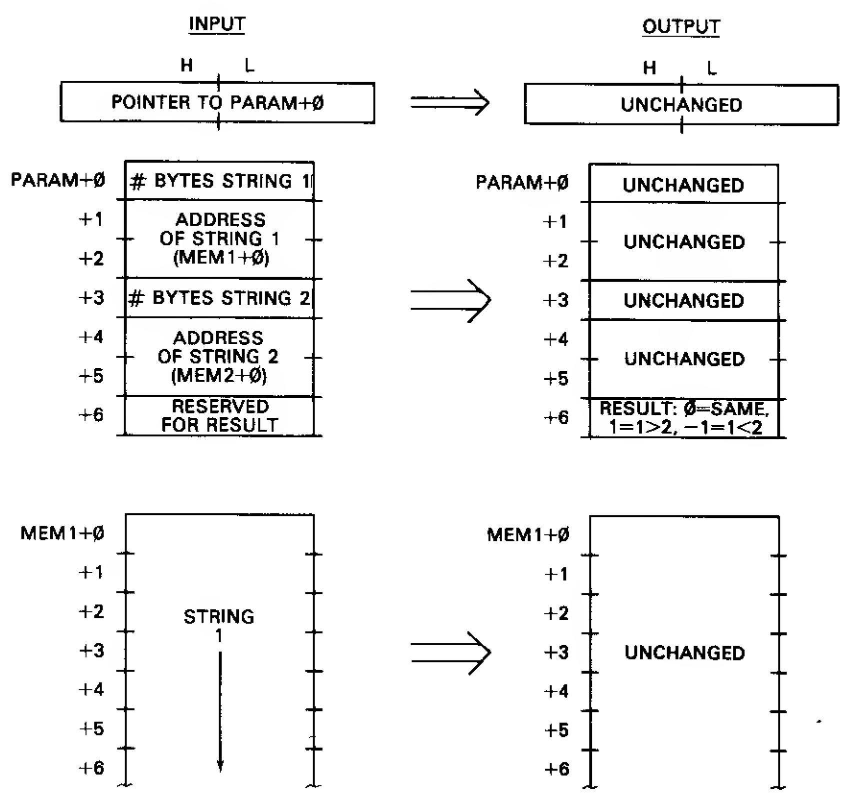
Description

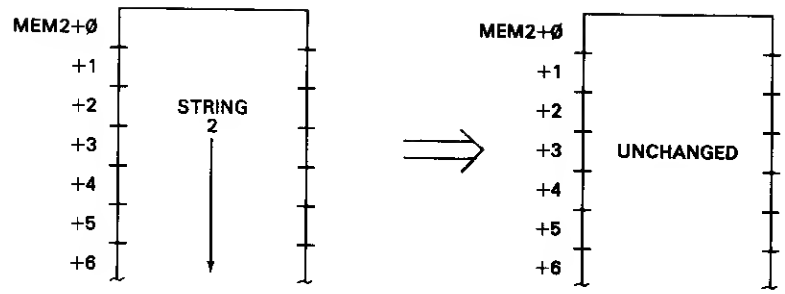
CSTRNG compares two strings and tests for equality, string 1 < string 2 and string 1 > string 2. By "string," we mean two blocks of memory that may or may not be of equal length containing byte-oriented data. This includes not only the BASIC definition of character strings, but other types of data as well, such as two strings of binary data. The comparison is an "unsigned" comparison where bytes in the range 080H through 0FFH are considered larger than zero.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first byte of the parameter block holds the number of bytes in string 1. The next two bytes contain the address of string 1 in standard Z-80 address format, least significant byte followed by most significant byte. The next byte in the parameter block holds the number of bytes in string 2. The next two bytes are the address of string 2 in Z-80 address format. The next byte of the parameter block (PARAM+6) is reserved for the result of the comparison.

On output, PARAM+6 holds a zero if the strings are equal, a minus number if string 1 < string 2, or a positive number if string 1 > string 2. For two strings of unequal length where the longer string holds the shorter string as a "substring," the result in PARAM+6 is negative if string 1 is shorter, or positive if string 2 is shorter.





Algorithm

The CSTRNG subroutine first compares the lengths of string 1 and string 2. It puts the smallest length value into the B register (CST010) and the comparison result of string 1 length—string 2 length in the C register.

Next, the address of string 2 is put into the IY register and the address of string 1 into the HL register.

The code at CST020 is the comparison loop. A subtract of each consecutive byte of the strings is done. Two conditions result from the subtract. If the subtracts are zero for the total number of bytes of the shorter string, the size comparison in C is put into the result. If this size comparison was zero, the strings are of equal length and are identical. If the size comparison was not zero, the comparison value reflects the "substring" condition detailed above.

If any subtract is not zero, the strings are unequal, and a jump to CST040 puts the sense of the comparison in the result.

Sample Calling Sequence

```
NAME OF SUBROUTINE? CSTRNG
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0   1   3        3 BYTES IN STRING 1
+ 1   2   45000    STRING 1 ADDRESS
+ 3   1   5        5 BYTES IN STRING 2
+ 4   2   46000    STRING 2 ADDRESS
+ 6   1   0
+ 7   0   0
MEMORY BLOCK 1 LOCATION? 45000
MEMORY BLOCK 1 VALUES?
+ 0   1   1   ┐
+ 1   1   255 ├STRING 1
+ 2   1   3   ┘
+ 3   0   0
MEMORY BLOCK 2 LOCATION? 46000
MEMORY BLOCK 2 VALUES?
+ 0   1   1   ┐
+ 1   1   254 │
+ 2   1   3   ├STRING 2
+ 3   1   4   │
+ 4   1   5   ┘
+ 5   0   0
MOVE SUBROUTINE TO? 38000
SUBROUTINE EXECUTED AT  38000
INPUT:               OUTPUT:
HL= 40000            HL= 40000
```

```
PARAM+ 0   3      PARAM+ 0   3
PARAM+ 1   200    PARAM+ 1   200
PARAM+ 2   175    PARAM+ 2   175
PARAM+ 3   5      PARAM+ 3   5      UNCHANGED
PARAM+ 4   176    PARAM+ 4   176
PARAM+ 5   179    PARAM+ 5   179
PARAM+ 6   0      PARAM+ 6   1    RESULT: STRING 1 > STRING 2
MEMB1+ 0   1      MEMB1+ 0   1
MEMB1+ 1   255    MEMB1+ 1   255
MEMB1+ 2   3      MEMB1+ 2   3
MEMB2+ 0   1      MEMB2+ 0   1
MEMB2+ 1   254    MEMB2+ 1   254    UNCHANGED
MEMB2+ 2   3      MEMB2+ 2   3
MEMB2+ 3   4      MEMB2+ 3   4
MEMB2+ 4   5      MEMB2+ 4   5

NAME OF SUBROUTINE?
```

Notes

1. The maximum number of bytes in either string may be 256, represented by 0 in the # of bytes parameter.
2. Output is a signed number at PARAM+6.

Program Listing

```
7F00                    ORG     7F00H           ;0520
               ;**********************************************************
               ;* STRING COMPARE. COMPARES TWO STRINGS.                  *
               ;*    INPUT: HL=> PARAMETER BLOCK                         *
               ;*           PARAM+0=# BYTES OF STRING 1                  *
               ;*           PARAM+1,+2=ADDRESS OF STRING 1               *
               ;*           PARAM+3=# BYTES OF STRING 2                  *
               ;*           PARAM+4,+5=ADDRESS OF STRING 2               *
               ;*           PARAM+6=RESERVED FOR RESULT                  *
               ;*    OUTPUT:PARAM+6=0 IF STRINGS EQUAL, - IF             *
               ;*           STRING1<STRING2, + IF STRING1>STRING2        *
               ;**********************************************************
               ;
7F00 F5        CSTRNG  PUSH    AF              ;SAVE REGISTERS
7F01 C5                PUSH    BC
7F02 E5                PUSH    HL
7F03 DDE5              PUSH    IX
7F05 FDE5              PUSH    IY
7F07 CD7F0A            CALL    0A7FH           ;***GET PB ADDRESS***
7F0A E5                PUSH    HL              ;TRANSFER TO IX
7F0B DDE1              POP     IX
7F0D DD4600            LD      B,(IX+0)        ;# OF 1
7F10 0E00              LD      C,0             ;STRING1=STRING 2 FLAG
7F12 DD7E00            LD      A,(IX+0)        ;GET # BYTES OF STRING 1
7F15 DDBE03            CP      (IX+3)          ;# OF 1-# OF 2
7F18 280B              JR      Z,CST010        ;GO IF STRINGS EQUAL LEN
7F1A 3807              JR      C,CST005        ;GO IF # OF 1<# OF 2
7F1C DD4603            LD      B,(IX+3)        ;GET SMALLER #
7F1F 0E01              LD      C,1             ;STRING 1>STRING 2
7F21 1802              JR      CST010
7F23 0EFF      CST005  LD      C,-1            ;STRING 1<STRING 2 CASE
7F25 DD6E04    CST010  LD      L,(IX+4)        ;GET ADDRESS OF STRING 2
7F28 DD6605            LD      H,(IX+5)
7F2B E5                PUSH    HL              ;TRANSFER TO IY
7F2C FDE1              POP     IY
7F2E DD6E01            LD      L,(IX+1)        ;GET ADDRESS OF STRING 1
```

```
7F31 DD6602   00460          LD      H,(IX+2)
7F34 7E       00470 CST020   LD      A,(HL)        ;GET STRING 1 BYTE
7F35 FD9600   00480          SUB     (IY+0)        ;COMPARE
7F38 2008     00490          JR      NZ,CST040     ;GO IF NOT EQUAL
7F3A 23       00500          INC     HL            ;BUMP STRING 1 POINTER
7F3B FD23     00510          INC     IY            ;BUMP STRING 2 POINTER
7F3D 10F5     00520          DJNZ    CST020        ;LOOP IF EQUAL
7F3F 79       00530          LD      A,C          ;GET SIZE COMPARISON
7F40 1806     00540          JR      CST050
7F42 3E01     00550 CST040   LD      A,1          ;STRING 1>STRING 2
7F44 3002     00560          JR      NC,CST050    ;GO IF OK
7F46 3EFF     00570          LD      A,-1         ;STRING 1<STRING 2
7F48 DD7706   00580 CST050   LD      (IX+6),A     ;STORE IN RESULT
7F4B FDE1     00590          POP     IY           ;RESTORE REGISTERS
7F4D DDE1     00600          POP     IX
7F4F E1       00610          POP     HL
7F50 C1       00620          POP     BC
7F51 F1       00630          POP     AF
7F52 C9       00640          RET                  ;RETURN TO CALLING PROGRAM
0000          00650          END
```

```
CSTRNG DECIMAL VALUES


245, 197, 229, 221, 229, 253, 229, 205, 127, 10,
229, 221, 225, 221, 70, 0, 14, 0, 221, 126,
0, 221, 190, 3, 40, 11, 56, 7, 221, 70,
3, 14, 1, 24, 2, 14, 255, 221, 110, 4,
221, 102, 5, 229, 253, 225, 221, 110, 1, 221,
102, 2, 126, 253, 150, 0, 32, 8, 35, 253,
35, 16, 245, 121, 24, 6, 62, 1, 48, 2,
62, 255, 221, 119, 6, 253, 225, 221, 225, 225,
193, 241, 201

CHKSUM= 55
```

DELBLK: DELETE BLOCK

System Configuration

Model I, Model III, Model II Stand Alone.

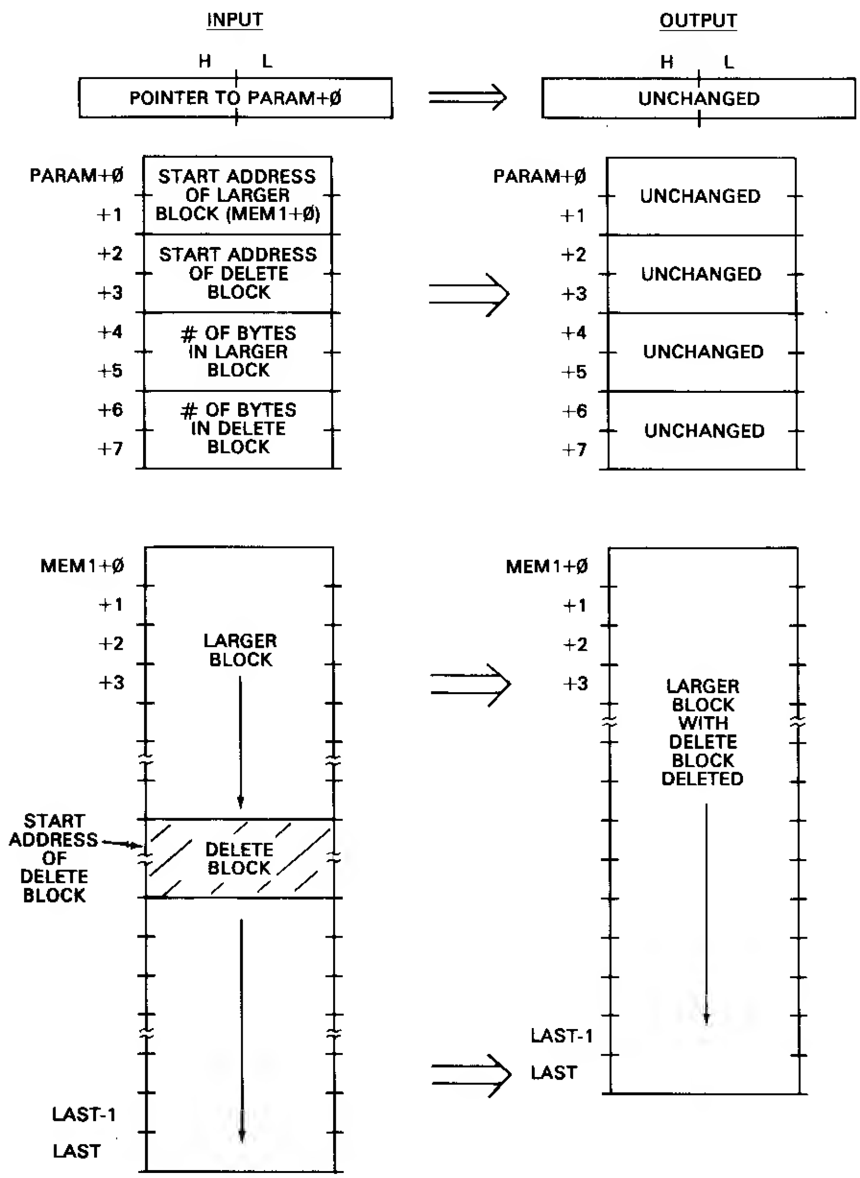
Description

DELBLK deletes a block in the middle of a larger block of memory. The block is deleted by moving up all bytes after the deletion block as shown below. This subroutine could be used for deleting a block of text, for example, and moving the remaining text into the deleted block. Both the "larger block" and "deletion block" may be any size up to the limits of memory.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the address of the larger block in standard Z-80 address format, least significant byte followed by most significant byte. The next two bytes are the address of the deletion block in Z-80 address format. The next two bytes of the parameter block (PARAM+4,+5) contain the number of bytes in the larger block; the next two bytes contain the number of bytes in the deletion block. Both are in standard Z-80 format.

On output, the contents of the parameter block remain unchanged. The deletion block has been deleted by a move of the remaining bytes of the larger block into the deletion area.



Algorithm

The DELBLK subroutine performs the deletion by doing a block move of the remaining bytes of the larger block into the deletion area. At the LDIR, HL contains the address of the location directly after the deletion block, DE contains the address of the deletion block, and BC contains the number of bytes remaining in the larger block after the deletion block.

The destination location (DE) is simply the deletion block address. This is saved for the LDIR in the stack. The source location (HL) is found by adding the deletion block address and the size of the deletion block. This is then pushed into the stack for LDIR use. The number to move is found by subtracting the source location (HL) from the last location of the larger block plus one.

Sample Calling Sequence

```
NAME OF SUBROUTINE? DELBLK
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0   2   45000    START OF LARGER BLOCK
+ 2   2   45003    START OF DELETION BLOCK
+ 4   2   10       10 BYTES IN LARGER BLOCK
+ 6   2   3        3 BYTES IN DELETION BLOCK
+ 8   0   0
MEMORY BLOCK 1 LOCATION? 45000
MEMORY BLOCK 1 VALUES?
+ 0   1   0   ─┐
+ 1   1   1    │
+ 2   1   2    │
+ 3   1   3 ┐  │
+ 4   1   4 ├ DELETION BLOCK  ├ LARGER BLOCK
+ 5   1   5 ┘  │
+ 6   1   6    │
+ 7   1   7    │
+ 8   1   8    │
+ 9   1   9   ─┘
+ 10  0   0
MEMORY BLOCK 2 LOCATION?
MOVE SUBROUTINE TO? 37777
SUBROUTINE EXECUTED AT  37777
INPUT:                OUTPUT:
HL= 40000             HL= 40000
PARAM+ 0   200        PARAM+ 0   200
PARAM+ 1   175        PARAM+ 1   175
PARAM+ 2   203        PARAM+ 2   203
PARAM+ 3   175        PARAM+ 3   175
PARAM+ 4   10         PARAM+ 4   10
PARAM+ 5   0          PARAM+ 5   0
PARAM+ 6   3          PARAM+ 6   3
PARAM+ 7   0          PARAM+ 7   0   ┐
MEMB1+ 0   0          MEMB1+ 0   0   │
MEMB1+ 1   1          MEMB1+ 1   1   │
MEMB1+ 2   2          MEMB1+ 2   2   │
MEMB1+ 3   3          MEMB1+ 3   6   ├ NEW BLOCK
MEMB1+ 4   4          MEMB1+ 4   7   │
MEMB1+ 5   5          MEMB1+ 5   8   │
MEMB1+ 6   6          MEMB1+ 6   9   ┘
MEMB1+ 7   7          MEMB1+ 7   7   ┐
MEMB1+ 8   8          MEMB1+ 8   8   ├ GARBAGE BYTES
MEMB1+ 9   9          MEMB1+ 9   9   ┘

NAME OF SUBROUTINE?
```

Notes

1. The maximum number of bytes in either block may be 65,535.
2. There will be a number of "garbage" bytes at the end of the larger block after the move.

```
7F00                      ORG     7F00H          ;0522
                  ;*****************************************************
                  ;* DELETE BLOCK. DELETES BLOCK IN MIDDLE OF LARGER BLOCK*
                  ;*    INPUT: HL=> PARAMETER BLOCK                     *
                  ;*           PARAM+0,+1=START ADDRESS OF LARGER BLOCK *
                  ;*           PARAM+2,+3=START ADDRESS OF DELETE BLOCK *
                  ;*           PARAM+4,+5=# OF BYTES IN LARGER BLOCK    *
                  ;*           PARAM+6,+7=# OF BYTES IN DELETE BLOCK    *
                  ;*    OUTPUT:DELETE BLOCK DELETED BY MOVING UP REMAIN- *
                  ;*           DER OF LARGER BLOCK                      *
                  ;*****************************************************
                  ;
7F00 C5           DELBLK  PUSH    BC             ;SAVE REGISTERS
7F01 D5                   PUSH    DE
7F02 E5                   PUSH    HL
7F03 DDE5                 PUSH    IX
7F05 CD7F0A               CALL    0A7FH          ;***GET PB ADDRESS***
7F08 E5                   PUSH    HL             ;TRANSFER TO IX
7F09 DDE1                 POP     IX
7F0B DD6E02               LD      L,(IX+2)       ;PUT DELETE BLK ADD IN HL
7F0E DD6603               LD      H,(IX+3)
7F11 E5                   PUSH    HL             ;DESTINATION FOR LDIR
7F12 DD4E06               LD      C,(IX+6)       ;PUT SIZE OF DEL BLK IN BC
7F15 DD4607               LD      B,(IX+7)
7F18 09                   ADD     HL,BC          ;FIND SOURCE LOC'N
7F19 E5                   PUSH    HL             ;SAVE FOR LDIR
7F1A DD6E00               LD      L,(IX+0)       ;PUT START INTO HL
7F1D DD6601               LD      H,(IX+1)
7F20 DD4E04               LD      C,(IX+4)       ;GET SIZE OF LARGE BLOCK
7F23 DD4605               LD      B,(IX+5)
7F26 09                   ADD     HL,BC          ;LAST LOC'N + ONE
7F27 D1                   POP     DE             ;GET SOURCE LOCATION
7F28 B7                   OR      A              ;CLEAR CARRY
7F29 ED52                 SBC     HL,DE          ;FIND # TO MOVE
7F2B E5                   PUSH    HL             ;TRANSFER TO BC
7F2C C1                   POP     BC
7F2D E1                   POP     HL             ;GET DESTINATION
7F2E EB                   EX      DE,HL          ;SWAP DE AND HL
7F2F EDB0                 LDIR                     ;MOVE 'EM
7F31 DDE1                 POP     IX             ;RESTORE REGISTERS
7F33 E1                   POP     HL
7F34 D1                   POP     DE
7F35 C1                   POP     BC
7F36 C9                   RET                    ;RETURN TO CALLING PROG
0000                      END
00000 TOTAL ERRORS
```

DELBLK DECIMAL VALUES

```
197, 213, 229, 221, 229, 205, 127, 10, 229, 221,
225, 221, 110, 2, 221, 102, 3, 229, 221, 78,
6, 221, 70, 7, 9, 229, 221, 110, 0, 221,
102, 1, 221, 78, 4, 221, 70, 5, 9, 209,
183, 237, 82, 229, 193, 225, 235, 237, 176, 221,
225, 225, 209, 193, 201
```

CHKSUM= 186

DRBOXS: DRAW BOX

System Configuration

Model I, Model III.

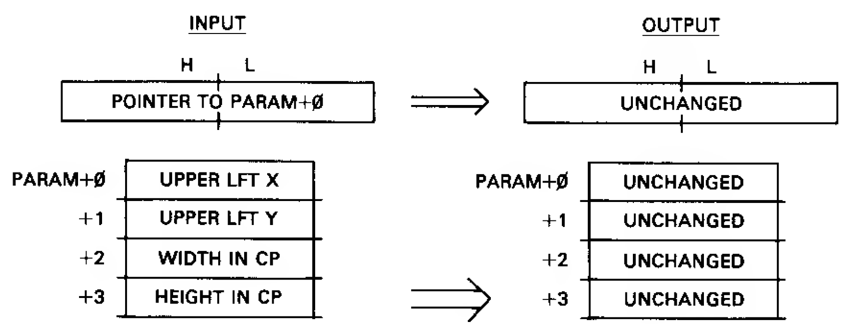
Description

DRBOXS draws a rectangle on the video display. The rectangle may start at any screen position and may be any size as long as it does not overrun the screen boundaries. The rectangle is drawn on a character position basis.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first byte of the parameter block contains the upper left-hand corner character position (x) from 0 to 63. The next byte of the parameter block contains the upper left-hand corner line position (y) from 0 to 15. The next byte of the parameter block contains the width of the rectangle in character positions, 2 to 63. The next byte of the parameter block contains the height of the rectangle in character positions, 2 to 16.

On output, the contents of the parameter block remain unchanged. The box has been drawn on the screen.



Algorithm

The DRBOXS subroutine contains two smaller subroutines called DRBWH and DRBWV. DRBWH draws a horizontal line, while DRBWV draws a vertical line. Both are not in the standard subroutine form because CALLs to the subroutine would not be relocatable.

DRBWH is entered from DRBOXS with HL containing the memory location that represents the leftmost character position for the horizontal line to be drawn, with B containing the width in character positions, and with C containing a flag for the return point.

DRBWV is entered from DRBOXS with HL containing the memory location that represents the topmost character position for the vertical line to be drawn, with B containing the height in character positions, and with C containing a flag for the return point.

In DRBOXS proper, there are four steps to draw the box. A call is made to DRBWH to draw the top line, a call is made to DRBWV to draw the right-hand line, a call is made to DRBWV to draw the left-hand line, and finally, a call is made to DRBWH to draw the bottom line.

First, the starting line position (y) is picked up and multiplied by 64 (DRB010). The result is added to the character position (x) and to the start of the screen location (3C00H). This result is the memory location representing the corner point. It is saved in the stack.

A call is then made to DRBWH to draw the top line. The return is made to DRB020.

HL now points to one location greater than the end of the line. HL is decremented and a call is made to DRBWV to draw the right-hand side. The return is made to DRB030.

The original corner location is now picked up from the stack, and a call is made to DRBWV to draw the left-hand line. The return is made to DRB040.

HL now points to one line greater than the bottom of the line. HL is decremented, and a call is made to DRBWH to draw the bottom line. The return is made to DRB050.

Sample Calling Sequence

```
NAME OF SUBROUTINE? DRBOXS
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0   1   32  ]
+ 1   1   8   ]  UPPER LEFT X, Y = 32, 8
+ 2   1   12     WIDTH = 12
+ 3   1   4      HEIGHT = 4
+ 4   0   0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 38888
SUBROUTINE EXECUTED AT  38888
INPUT:                OUTPUT:
HL= 40000             HL= 40000
PARAM+ 0   32         PARAM+ 0   32 ]
PARAM+ 1   8          PARAM+ 1   8  ]
PARAM+ 2   12         PARAM+ 2   12 ]  UNCHANGED
PARAM+ 3   4          PARAM+ 3   4  ]
```

Notes

1. If the parameters cause the rectangle to exceed screen limits, the system may be "bombed."
2. The top and bottom lines are wider than the side lines in the rectangle.

Program Listing

```
7F00                   ORG     7F00H           ;0522
        ;*****************************************************
        ;* DRAW BOX. DRAWS BOX OF GIVEN WIDTH AND HEIGHT AT  *
        ;* SPECIFIED LOCATION.                               *
        ;*     INPUT: HL=> PARAMETER BLOCK                   *
        ;*            PARAM+0=UPPER LEFT CORNER CHAR POS (X) *
        ;*            PARAM+1=UPPER LEFT CORNER LINE # (Y)   *
        ;*            PARAM+2=WIDTH IN CHARACTER POSITIONS   *
        ;*            PARAM+3=HEIGHT IN CHARACTER POSITIONS  *
        ;*     OUTPUT:BOX DRAWN ON SCREEN                    *
        ;*****************************************************
        ;
```

```
7F00 C5          00220 DRBOXS  PUSH    BC              ;SAVE REGISTERS
7F01 D5          00230         PUSH    DE
7F02 E5          00240         PUSH    HL
7F03 DDE5        00250         PUSH    IX
7F05 CD7F0A      00260         CALL    0A7FH           ;***GET PB LOC'N***
7F08 E5          00270         PUSH    HL              ;TRANSFER TO IX
7F09 DDE1        00280         POP     IX
7F0B DD6E01      00290         LD      L,(IX+1)        ;GET Y IN LINES
7F0E 2600        00300         LD      H,0             ;NOW IN HL
7F10 0606        00310         LD      B,6             ;ITERATION COUNT
7F12 29          00320 DRB010  ADD     HL,HL             ;FIND LINE DISPLACEMENT
7F13 10FD        00330         DJNZ    DRB010            ;LINE # * 64
7F15 DD4E00      00340         LD      C,(IX+0)        ;GET CHAR POSITION
7F18 0600        00350         LD      B,0             ;NOW IN BC
7F1A 09          00360         ADD     HL,BC           ;FIND DISPL FROM START
7F1B 01003C      00370         LD      BC,3C00H        ;START OF SCREEN
7F1E 09          00380         ADD     HL,BC           ;FIND ACTUAL MEMORY LOC'N
7F1F E5          00390         PUSH    HL              ;SAVE LOC'N
7F20 DD4602      00400         LD      B,(IX+2)        ;GET WIDTH IN CHAR POSNS
7F23 0E00        00410         LD      C,0             ;FLAG FOR RETURN
7F25 181C        00420         JR      DRBWH           ;DRAW TOP LINE
7F27 2B          00430 DRB020  DEC     HL              ;POINT TO END OF LINE
7F28 DD4603      00440         LD      B,(IX+3)        ;GET HEIGHT IN CHAR POSNS
7F2B 1821        00450         JR      DRBWV           ;DRAW RIGHT SIDE
7F2D E1          00460 DRB030  POP     HL              ;GET UPPER LEFT CORNER LOC
7F2E DD4603      00470         LD      B,(IX+3)        ;GET HEIGHT IN CHAR POSNS
7F31 0E01        00480         LD      C,1             ;FLAG FOR RETURN
7F33 1819        00490         JR      DRBWV           ;DRAW LEFT SIDE
7F35 B7          00500 DRB040  OR      A               ;CLEAR CARRY
7F36 ED52        00510         SBC     HL,DE           ;POINT TO END OF LINE
7F38 DD4602      00520         LD      B,(IX+2)        ;GET WIDTH IN CHAR POSNS
7F3B 1806        00530         JR      DRBWH           ;DRAW BOTTOM LINE
7F3D DDE1        00540 DRB050  POP     IX              ;RESTORE REGISTERS
7F3F E1          00550         POP     HL
7F40 D1          00560         POP     DE
7F41 C1          00570         POP     BC
7F42 C9          00580         RET                     ;RETURN TO CALLING PROG
7F43 36BF        00590 DRBWH   LD      (HL),0BFH         ;SET CHAR POSN TO ALL ON
7F45 23          00600         INC     HL                ;HORIZ INCREMENT
7F46 10FB        00610         DJNZ    DRBWH             ;LOOP 'TIL LINE DONE
7F48 CB41        00620         BIT     0,C             ;TEST FLAG
7F4A 28DB        00630         JR      Z,DRB020        ;RTN POINT 1
7F4C 18EF        00640         JR      DRB050          ;RTN POINT 2
7F4E 114000      00650 DRBWV   LD      DE,40H          ;INCREMENT FOR VERTICAL LN
7F51 36BF        00660 DRBWV1  LD      (HL),0BFH         ;SET CHAR POSN TO ALL ON
7F53 19          00670         ADD     HL,DE             ;POINT TO NEXT POSITION
7F54 10FB        00680         DJNZ    DRBWV1            ;LOOP 'TIL LINE DONE
7F56 CB41        00690         BIT     0,C             ;TEST FLAG
7F58 28D3        00700         JR      Z,DRB030        ;RTN POINT 1
7F5A 18D9        00710         JR      DRB040          ;RTN POINT 2
0000             00720         END
00000 TOTAL ERRORS
```

```
DRBOXS DECIMAL VALUES

197, 213, 229, 221, 229, 205, 127, 10, 229, 221,
225, 221, 110, 1, 38, 0, 6, 6, 41, 16,
253, 221, 78, 0, 6, 0, 9, 1, 0, 60,
9, 229, 221, 70, 2, 14, 0, 24, 28, 43,
221, 70, 3, 24, 33, 225, 221, 70, 3, 14,
1, 24, 25, 183, 237, 82, 221, 70, 2, 24,
6, 221, 225, 225, 209, 193, 201, 54, 191, 35,
16, 251, 203, 65, 40, 219, 24, 239, 17, 64,
0, 54, 191, 25, 16, 251, 203, 65, 40, 211,
24

CHKSUM= 128
```

Configuration

Model I, Model III.

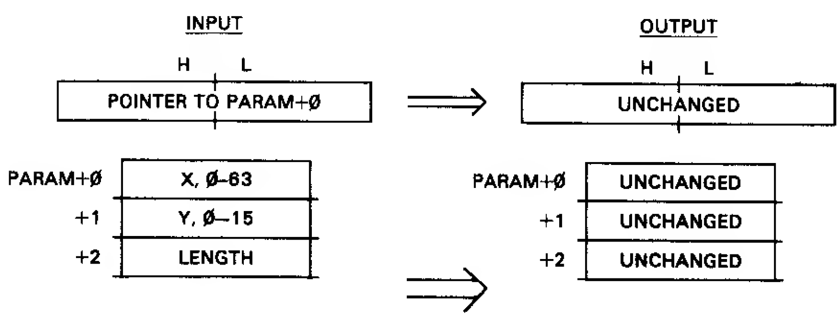
Description

DRHLNE draws a horizontal line on the screen. The line may be any length and may start on any character position of any screen line.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first byte of the parameter block contains the starting x character position of the line, from 0 to 63. The leftmost character position of the line must be specified. The next byte of the parameter block contains the starting line number y of the line, from 0 to 15. The next byte of the parameter block contains the number of character positions in the line length. This will be a maximum of 64 for a line that starts at the left edge of the screen.

On output, the parameter block contents are unchanged. The horizontal line has been drawn.



Algorithm

The DRHLNE subroutine performs the move by computing the starting address of the line in video display memory and by controlling the operation with the count of the number of character positions involved.

First, the line number value is picked up from the parameter block. This is multiplied by 64 to find the number of bytes (displacement) from the start of video display memory. This value is added to 3C00H to find the actual video memory address for the line start. This value is added to the character position of the start from the parameter block to find the starting position in video display memory.

A byte of 0BFH is stored for each character position in the line. The current video display memory position in HL is then incremented to find the next location of the line. A count of the number of character positions involved is then decremented and a jump is made to DRH020 if the count is not zero.

Sample Calling Sequence

```
NAME OF SUBROUTINE? DRHLNE
HL VALUE? 50000
PARAMETER BLOCK LOCATION? 50000
PARAMETER BLOCK VALUES?
+ 0  1  0
+ 1  1  15     X, Y = 0, 15
+ 2  1  64     LENGTH = 64
+ 3  0  0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 45000
SUBROUTINE EXECUTED AT  45000
INPUT:              OUTPUT:
HL= 50000           HL= 50000
PARAM+ 0  0         PARAM+ 0  0
PARAM+ 1  15        PARAM+ 1  15   UNCHANGED
PARAM+ 2  64        PARAM+ 2  64

NAME OF SUBROUTINE?
```

Notes

1. The program may "bomb" the system if the length of travel goes beyond video display memory boundaries.
2. The program may "bomb" the system if the x and y coordinates are improperly specified.
3. Change location 7F22H to draw a narrower line.

Program Listing

```
7F00                          ORG     7F00H           ;0522
                    ;*****************************************************
                    ;* DRAW HORIZONTAL LINE. DRAWS A HORIZONTAL LINE FROM  *
                    ;* GIVEN LINE (Y), CHARACTER POSITION (X).             *
                    ;*    INPUT: HL=> PARAMETER BLOCK                      *
                    ;*           PARAM+0=CHAR POSITION (X), 0 - 63         *
                    ;*           PARAM+1=LINE NUMBER (Y), 0-15             *
                    ;*           PARAM+2=LENGTH OF LINE IN CHAR POSITIONS  *
                    ;*    OUTPUT:LINE DRAWN                                *
                    ;*****************************************************
                    ;
7F00 C5             DRHLNE  PUSH    BC              ;SAVE REGISTERS
7F01 E5                     PUSH    HL
7F02 DDE5                   PUSH    IX
7F04 CD7F0A                 CALL    0A7FH           ;***GET PB LOC'N***
7F07 E5                     PUSH    HL              ;TRANSFER TO IX
7F08 DDE1                   POP     IX
7F0A DD6E01                 LD      L,(IX+1)        ;GET LINE NUMBER
7F0D 2600                   LD      H,0             ;NOW IN HL
7F0F 0606                   LD      B,6             ;ITERATION COUNT
7F11 29             DRH010  ADD     HL,HL             ;MULTIPLY LINE # * 64
7F12 10FD                   DJNZ    DRH010            ;LOOP TILL DONE
7F14 DD4E00                 LD      C,(IX+0)        ;GET CHAR POS'N (X)
7F17 0600                   LD      B,0             ;NOW IN BC
7F19 09                     ADD     HL,BC           ;DISPLACEMENT FROM START
7F1A 01003C                 LD      BC,3C00H        ;START OF SCREEN
7F1D 09                     ADD     HL,BC           ;FIND ACTUAL START LOC'N
7F1E DD4602                 LD      B,(IX+2)        ;GET NUMBER OF CHAR POS'NS
7F21 36BF           DRH020  LD      (HL),0BFH         ;ALL ON FOR CHAR POSITION
7F23 23                     INC     HL                ;BUMP POINTER
```

```
7F24 10FB        00400         DJNZ     DRH020          ;LOOP 'TIL DONE
7F26 DDE1        00410         POP      IX             ;RESTORE REGISTERS
7F28 E1          00420         POP      HL
7F29 C1          00430         POP      BC
7F2A C9          00440         RET                     ;RETURN TO CALLING PROG
0000             00450         END
00000 TOTAL ERRORS
```

```
DRHLNE DECIMAL VALUES


 197, 229, 221, 229, 205, 127, 10, 229, 221, 225,
 221, 110, 1, 38, 0, 6, 6, 41, 16, 253,
 221, 78, 0, 6, 0, 9, 1, 0, 60, 9,
 221, 70, 2, 54, 191, 35, 16, 251, 221, 225,
 225, 193, 201

CHKSUM= 10
```

DRVLNE: DRAW VERTICAL LINE

Configuration

Model I, Model III.

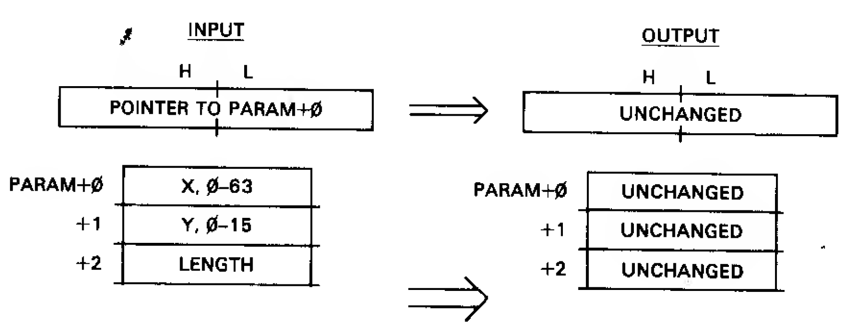
Description

DRVLNE draws a vertical line on the screen. The line may be any length and may start on any character position of any screen line.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first byte of the parameter block contains the starting x character position of the line, from 0 to 63. The topmost character position of the line must be specified. The next byte of the parameter block contains the starting line number y of the line, from 0 to 15. The next byte of the parameter block contains the number of character positions in the line length. This will be a maximum of 16 for a line that starts at the top of the screen.

On output, the parameter block contents are unchanged. The vertical line has been drawn.



Algorithm

The DRVLNE subroutine performs the move by computing the starting address of the line in video display memory and by controlling the operation with the count of the number of character positions involved.

First, the line number value is picked up from the parameter block. This is multiplied by 64 to find the number of bytes (displacement) from the start of video display memory. This value is added to a character position of the start from the parameter block to find the displacement from the start of video display memory. This value is added to 3C00H to find the actual video memory address for the line start.

A byte of 0BFH is stored for each character position in the line. The current video display memory position in HL is then incremented by 40H to find the next location of the line. A count of the number of character positions involved is then decremented and a jump is made to DRV020 if the count is not zero.

Sample Calling Sequence

```
NAME OF SUBROUTINE? DRVLNE
HL VALUE? 50000
PARAMETER BLOCK LOCATION? 50000
PARAMETER BLOCK VALUES?
+ 0   1   8  } X, Y = 8, 9
+ 1   1   9  }
+ 2   1   5    LENGTH = 5
+ 3   0   0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 40100
SUBROUTINE EXECUTED AT  40100
INPUT:               OUTPUT:
HL= 50000            HL= 50000
PARAM+ 0   8         PARAM+ 0   8 }
PARAM+ 1   9         PARAM+ 1   9 } UNCHANGED
PARAM+ 2   5         PARAM+ 2   5 }

NAME OF SUBROUTINE?
```

Notes

1. The program may "bomb" the system if the length of travel goes beyond video display memory boundaries.

2. The program may "bomb" the system if the x and y coordinates are improperly specified.

Program Listing

```
7F00            ORG     7F00H           ;0522
        ;***********************************************************
        ;* DRAW VERTICAL LINE. DRAWS A VERTICAL LINE FROM          *
        ;* GIVEN LINE (Y), CHARACTER POSITION (X).                 *
        ;*    INPUT: HL=> PARAMETER BLOCK                          *
        ;*            PARAM+0=CHAR POSITION (X), 0 - 63            *
        ;*            PARAM+1=LINE NUMBER (Y), 0-15                *
        ;*            PARAM+2=LENGTH OF LINE IN CHAR POSITIONS     *
        ;*    OUTPUT:LINE DRAWN                                    *
        ;***********************************************************
        ;
```

```
7F00 C5        00210 DRVLNE  PUSH    BC              ;SAVE REGISTERS
7F01 D5        00220         PUSH    DE
7F02 E5        00230         PUSH    HL
7F03 DDE5      00240         PUSH    IX
7F05 CD7F0A    00250         CALL    0A7FH           ;***GET PB LOC'N***
7F08 E5        00260         PUSH    HL              ;TRANSFER TO IX
7F09 DDE1      00270         POP     IX
7F0B DD6E01    00280         LD      L,(IX+1)        ;GET LINE NUMBER
7F0E 2600      00290         LD      H,0             ;NOW IN HL
7F10 0606      00300         LD      B,6             ;ITERATION COUNT
7F12 29        00310 DRV010  ADD     HL,HL             ;MULTIPLY LINE # * 64
7F13 10FD      00320         DJNZ    DRV010            ;LOOP TILL DONE
7F15 DD4E00    00330         LD      C,(IX+0)        ;GET CHAR POS'N (X)
7F18 0600      00340         LD      B,0             ;NOW IN BC
7F1A 09        00350         ADD     HL,BC           ;DISPLACEMENT FROM START
7F1B 01003C    00360         LD      BC,3C00H        ;START OF SCREEN
7F1E 09        00370         ADD     HL,BC           ;FIND ACTUAL START LOC'N
7F1F DD4602    00380         LD      B,(IX+2)        ;GET NUMBER OF CHAR POSNS
7F22 114000    00390         LD      DE,40H          ;LINE DISPLACEMENT
7F25 36BF      00400 DRV020  LD      (HL),0BFH         ;ALL ON FOR CHAR POSITION
7F27 19        00410         ADD     HL,DE             ;FIND NEXT POSITION
7F28 10FB      00420         DJNZ    DRV020            ;LOOP 'TIL DONE
7F2A DDE1      00430         POP     IX              ;RESTORE REGISTERS
7F2C E1        00440         POP     HL
7F2D D1        00450         POP     DE
7F2E C1        00460         POP     BC
7F2F C9        00470         RET                     ;RETURN TO CALLING PROG
0000           00480         END
00000 TOTAL ERRORS
```

```
DRVLNE DECIMAL VALUES

197, 213, 229, 221, 229, 205, 127, 10, 229, 221,
225, 221, 110, 1, 38, 0, 6, 6, 41, 16,
253, 221, 78, 0, 6, 0, 9, 1, 0, 60,
9, 221, 70, 2, 17, 64, 0, 54, 191, 25,
16, 251, 221, 225, 225, 209, 193, 201

CHKSUM= 247
```

DSEGHT: DIVIDE 16 BY 8

System Configuration

Model I, Model III, Model II Stand Alone.

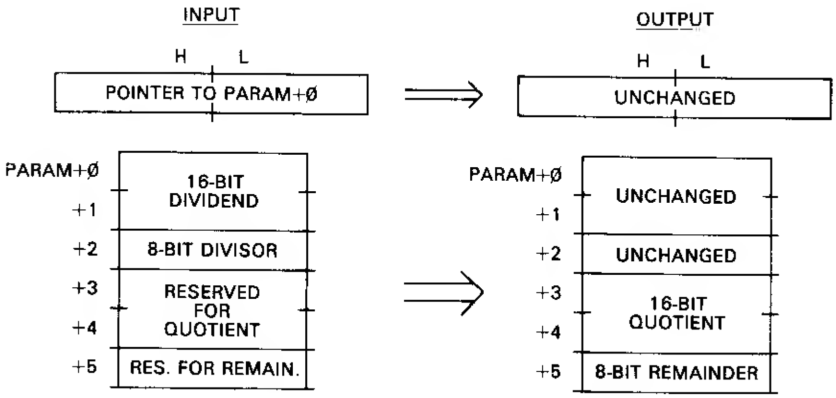
Description

DSEGHT divides a 16-bit binary number by an 8-bit binary number. The divide is an "unsigned" divide, where both numbers are considered to be absolute numbers without sign. Both the quotient and remainder are returned.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the 16-bit dividend. The next byte of the parameter block contains an 8-bit divisor. The next two bytes of the parameter block are reserved for the 16-bit quotient. The next byte is reserved for the 8-bit remainder.

On output, PARA+3, +4 hold the 16-bit quotient and PARA+5 holds the 8-bit remainder. The contents of the rest of the parameter block remain unchanged.



Algorithm

The DSEGHT subroutine performs the divide by a "restoring" type of bit-by-bit binary divide. The dividend is put into the HL register pair. The divisor is put into the C register. The A register is cleared. For each of 16 iterations in the divide, the HL register pair is shifted left one bit position into the A register. A subtract of the divisor (C) from the "residue" in A is then done. If the result is positive, a one bit is put into the least significant bit of HL. If the result is negative, a zero bit is put into the least significant bit of HL, and the previous value in A is restored by an add.

Quotient bits fill up the HL register from the right as the residue is shifted out into the A register toward the left. At the end of 16 iterations, the HL register pair contains the 16 quotient bits and the A register contains an 8-bit remainder.

The code at DSE010 is the main loop in DSEGHT which shifts HL left by an "ADD HL,HL" and "ADC A,A." The lsb of HL is preset with a quotient bit of one, and the subtract of C from A is done. If the result is positive, a loop to DSE010 is done for the next iteration. If the result is negative, C is added back to A, and the lsb of HL is reset. The B register holds the iteration count.

Sample Calling Sequence

```
NAME OF SUBROUTINE? DSEGHT
HL VALUE? 42200
PARAMETER BLOCK LOCATION? 42200
PARAMETER BLOCK VALUES?
+ 0  2  60000  DIVIDEND
+ 2  1  111    DIVISOR
+ 3  2  0
+ 5  1  0
+ 6  0  0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 43000
SUBROUTINE EXECUTED AT  43000
INPUT:             OUTPUT:
HL= 42200          HL= 42200
```

```
PARAM+ 0   96      PARAM+ 0   96  ┐
PARAM+ 1   234     PARAM+ 1   234 ├UNCHANGED
PARAM+ 2   111     PARAM+ 2   111 ┘
PARAM+ 3   0       PARAM+ 3   28  ┐
PARAM+ 4   0       PARAM+ 4   2   ┘QUOTIENT = 540
PARAM+ 5   0       PARAM+ 5   60   REMAINDER = 60

NAME OF SUBROUTINE?
```

Notes

1. Maximum dividend is 65,535. Maximum divisor is 255. The maximum quotient will be 65,535 and the maximum remainder will be 255.
2. Division by 0 causes an invalid result of 0FFFFH.

Program Listing

```
7F00          00100          ORG     7F00H           ;0522
              00110 ;***********************************************************
              00120 ;* DIVIDE 16 BY 8. DIVIDES A 16-BIT UNSIGNED NUMBER BY    *
              00130 ;* AN 8-BIT UNSIGNED NUMBER TO GIVE A QUOTIENT AND RE-   *
              00140 ;* MAINDER.                                               *
              00150 ;*     INPUT: HL=> PARAMETER BLOCK                        *
              00160 ;*            PARAM+0,+1=16-BIT DIVIDEND                  *
              00170 ;*            PARAM+2=8-BIT DIVISOR                       *
              00180 ;*            PARAM+3,+4=RESERVED FOR QUOTIENT            *
              00190 ;*            PARAM+5=RESERVED FOR REMAINDER              *
              00200 ;*    OUTPUT:PARAM+3,+4 HOLDS 16-BIT QUOTIENT             *
              00210 ;*            PARAM+5 HOLDS 8-BIT REMAINDER               *
              00220 ;***********************************************************
              00230 ;
7F00 F5       00240 DSEGHT   PUSH    AF              ;SAVE REGISTERS
7F01 C5       00250          PUSH    BC
7F02 E5       00260          PUSH    HL
7F03 DDE5     00270          PUSH    IX
7F05 CD7F0A   00280          CALL    0A7FH           ;***GET PB LOC'N***
7F08 E5       00290          PUSH    HL              ;TRANSFER TO IX
7F09 DDE1     00300          POP     IX
7F0B 0610     00310          LD      B,16            ;ITERATION COUNT
7F0D DD4E02   00320          LD      C,(IX+2)        ;LOAD DIVISOR
7F10 DD6E00   00330          LD      L,(IX+0)        ;PUT DIVIDEND IN HL
7F13 DD6601   00340          LD      H,(IX+1)
7F16 AF       00350          XOR     A               ;CLEAR EXTENSION REG
7F17 29       00360 DSE010   ADD     HL,HL             ;SHIFT HL LEFT 1 BIT
7F18 8F       00370          ADC     A,A               ;SHIFT A LEFT W/CARRY
7F19 2C       00380          INC     L                 ;SET Q BIT TO 1
7F1A 91       00390          SUB     C                 ;SUBTRACT D'SOR FROM D'END
7F1B 3002     00400          JR      NC,DSE020         ;GO IF SUBTRACT WENT
7F1D 81       00410          ADD     A,C               ;RESTORE
7F1E 2D       00420          DEC     L                 ;RESET Q BIT
7F1F 10F6     00430 DSE020   DJNZ    DSE010            ;LOOP FOR 16 ITERATIONS
7F21 DD7503   00440          LD      (IX+3),L        ;STORE QUOTIENT
7F24 DD7404   00450          LD      (IX+4),H
7F27 DD7705   00460          LD      (IX+5),A        ;STORE REMAINDER
7F2A DDE1     00470          POP     IX              ;RESTORE REGISTERS
7F2C E1       00480          POP     HL
7F2D C1       00490          POP     BC
7F2E F1       00500          POP     AF
7F2F C9       00510          RET                     ;RETURN TO CALLING PROG
0000          00520          END
00000 TOTAL ERRORS
```

```
DSEGHT DECIMAL VALUES

 245, 197, 229, 221, 229, 2Ø5, 127, 1Ø, 229, 221,
 225, 6, 16, 221, 78, 2, 221, 11Ø, Ø, 221,
 1Ø2, 1, 175, 41, 143, 44, 145, 48, 2, 129,
 45, 16, 246, 221, 117, 3, 221, 116, 4, 221,
 119, 5, 221, 225, 225, 193, 241, 2Ø1

CHKSUM= 83
```

DSSIXT: DIVIDE 16 BY 16

System Configuration

Model I, Model III, Model II Stand Alone.

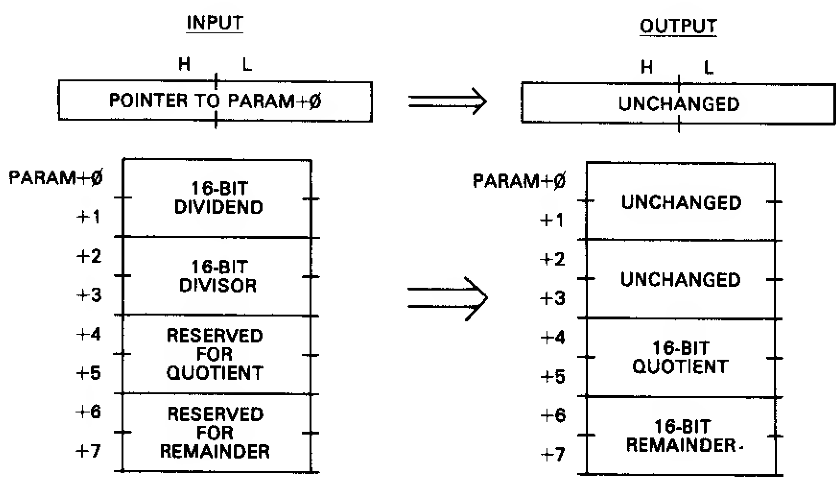
Description

DSSIXT divides a 16-bit binary number by a 16-bit binary number. The divide is an "unsigned" divide, where both numbers are considered to be absolute numbers without sign. Both the quotient and remainder are returned.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the 16-bit dividend. The next two bytes of the parameter block contain a 16-bit divisor. The next two bytes of the parameter block are reserved for the 16-bit quotient. The next two bytes are reserved for the 16-bit remainder.

On output, PARA+4, +5 hold the 16-bit quotient and PARA+6, +7 holds the 8-bit remainder. The contents of the rest of the parameter block remain unchanged.



Algorithm

The DSEGHT subroutine performs the divide by a "restoring" type of bit-by-bit binary divide. The dividend is put into the DE register pair. The divisor is put into the BC register pair. The HL register is cleared. For each of 16 iterations in the divide, the DE register pair is shifted left one bit position into the HL register pair. A subtract of the divisor (BC) from the "residue" in HL is then done. If the result is positive, a one bit is put into the least significant bit of DE. If the result is negative, a zero bit is put into the least significant bit of DE, and the previous value in HL is restored by an add.

Quotient bits fill up the DE register from the right as the residue is shifted out into the HL register pair toward the left. At the end of 16 iterations, the DE register pair contains the 16 quotient bits and the HL register contains a 16-bit remainder.

The code at DSS020 is the main loop in DSSIXT which shifts DE left by an exchange of DE and HL, an "ADD HL,HL," and an exchange back. HL is shifted by an "ADC HL,HL," merging any carry from DE. The lsb of DE is preset with a quotient bit of one, and the subtract of BC from HL is done. If the result is positive, a loop is made back to DSS020 for the next iteration. If the result is negative, BC is added back to HL, and the lsb of DE is reset. The A register holds the iteration count.

Sample Calling Sequence

```
NAME OF SUBROUTINE? DSSIXT
HL VALUE? 45000
PARAMETER BLOCK LOCATION? 45000
PARAMETER BLOCK VALUES?
+ 0  2  10000  DIVIDEND
+ 2  2  999    DIVISOR
+ 4  2  0
+ 6  2  0
+ 8  0  0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 50000
SUBROUTINE EXECUTED AT  50000
INPUT:             OUTPUT:
HL= 45000          HL= 45000
PARAM+ 0  16       PARAM+ 0  16  ┐
PARAM+ 1  39       PARAM+ 1  39  │
PARAM+ 2  231      PARAM+ 2  231 ├UNCHANGED
PARAM+ 3  3        PARAM+ 3  3   ┘
PARAM+ 4  0        PARAM+ 4  10  ┐
PARAM+ 5  0        PARAM+ 5  0   ┘QUOTIENT = 10
PARAM+ 6  0        PARAM+ 6  10  ┐
PARAM+ 7  0        PARAM+ 7  0   ┘REMAINDER = 0

NAME OF SUBROUTINE?
```

Notes

1. Maximum dividend is 65,535. Maximum divisor is 65,535. The maximum quotient will be 65,535 and the maximum remainder will be 65,535.

2. Division by 0 causes an invalid result of 0FFFFH.

```
7F00          00100         ORG     7F00H              ;0522
              00110 ;*****************************************************
              00120 ;* DIVIDE 16 BY 16. DIVIDES A 16-BIT UNSIGNED NUMBER BY *
              00130 ;* A 16-BIT UNSIGNED NUMBER TO GIVE A QUOTIENT AND RE-  *
              00140 ;* MAINDER.                                              *
              00150 ;*    INPUT: HL=> PARAMETER BLOCK                         *
              00160 ;*           PARAM+0,+1=16-BIT DIVIDEND                   *
              00170 ;*           PARAM+2,+3=16-BIT DIVISOR                    *
              00180 ;*           PARAM+4,+5=RESERVED FOR QUOTIENT             *
              00190 ;*           PARAM+6,+7=RESERVED FOR REMAINDER            *
              00200 ;*    OUTPUT:PARAM+4,+5 HOLDS 16-BIT QUOTIENT             *
              00210 ;*           PARAM+6,+7 HOLDS 16-BIT REMAINDER            *
              00220 ;*****************************************************
              00230 ;
7F00 F5       00240 DSSIXT  PUSH    AF                 ;SAVE REGISTERS
7F01 C5       00250         PUSH    BC
7F02 D5       00260         PUSH    DE
7F03 E5       00270         PUSH    HL
7F04 DDE5     00280         PUSH    IX
7F06 CD7F0A   00290         CALL    0A7FH              ;***GET PB LOC'N***
7F09 E5       00300         PUSH    HL                 ;TRANSFER TO IX
7F0A DDE1     00310         POP     IX
7F0C DD5E00   00320         LD      E,(IX+0)           ;PUT DIVIDEND INTO DE
7F0F DD5601   00330         LD      D,(IX+1)
7F12 DD4E02   00340         LD      C,(IX+2)           ;PUT DIVISOR INTO BC
7F15 DD4603   00350         LD      B,(IX+3)
7F18 210000   00360         LD      HL,0               ;ZERO HL
7F1B 3E10     00370         LD      A,16               ;ITERATION COUNT
7F1D EB       00380 DSS020  EX      DE,HL                ;DE TO HL
7F1E 29       00390         ADD     HL,HL                ;SHIFT LEFT
7F1F EB       00400         EX      DE,HL                ;DE BACK
7F20 ED6A     00410         ADC     HL,HL                ;SHIFT LEFT PLUS CARRY
7F22 13       00420         INC     DE                   ;SET Q BIT TO 1
7F23 B7       00430         OR      A                    ;CLEAR CARRY
7F24 ED42     00440         SBC     HL,BC                ;SUB DIVISOR FROM DIVIDEND
7F26 3002     00450         JR      NC,DSS030            ;GO IF SUBTRACT OK
7F28 1B       00460         DEC     DE                   ;RESET Q BIT
7F29 09       00470         ADD     HL,BC                ;RESTORE
7F2A 3D       00480 DSS030  DEC     A                    ;DECREMENT ITERATION CNT
7F2B 20F0     00490         JR      NZ,DSS020            ;LOOP FOR 16 ITERATIONS
7F2D DD7304   00500         LD      (IX+4),E           ;STORE QUOTIENT
7F30 DD7205   00510         LD      (IX+5),D
7F33 DD7506   00520         LD      (IX+6),L           ;STORE REMAINDER
7F36 DD7407   00530         LD      (IX+7),H
7F39 DDE1     00540         POP     IX                 ;RESTORE REGISTERS
7F3B E1       00550         POP     HL
7F3C D1       00560         POP     DE
7F3D C1       00570         POP     BC
7F3E F1       00580         POP     AF
7F3F C9       00590         RET                        ;RETURN TO CALLING PROG
0000          00600         END
00000 TOTAL ERRORS
```

```
DSSIXT DECIMAL VALUES

 245, 197, 213, 229, 221, 229, 205, 127, 10, 229,
 221, 225, 221, 94, 0, 221, 86, 1, 221, 78,
 2, 221, 70, 3, 33, 0, 0, 62, 16, 235,
 41, 235, 237, 106, 19, 183, 237, 66, 48, 2,
 27, 9, 61, 32, 240, 221, 115, 4, 221, 114,
 5, 221, 117, 6, 221, 116, 7, 221, 225, 225,
 209, 193, 241, 201

CHKSUM= 149
```

System Configuration

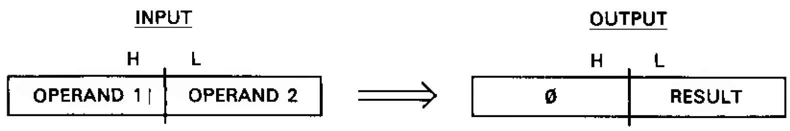
Model I, Model III, Model II Stand Alone.

Description

EXCLOR performs an exclusive OR on two 8-bit operands.

Input/Output Parameters

On input, the H register contains operand number one and the L register contains operand number two. On output, L contains the 8-bit result.



Algorithm

The EXCLOR subroutine performs the exclusive OR by the XOR instruction and returns the result in the L register with H set to zero.

Sample Calling Sequence

```
NAME OF SUBROUTINE? EXCLOR
HL VALUE? 13141  H = 51 = 00110011; L = 85 = 01010101
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 41111
SUBROUTINE EXECUTED AT  41111
INPUT:             OUTPUT:
HL= 13141          HL= 102  RESULT: 00110011 XOR 01010101 = 01100110

NAME OF SUBROUTINE?
```

Notes

1. BASIC contains no exclusive OR command.

Program Listing

```
7F00           00100          ORG     7F00H          ;0522
               00110 ;*****************************************************
               00120 ;* EXCLUSIVE OR. PERFORMS EXCLUSIVE OR OF TWO EIGHT-BIT *
               00130 ;* OPERANDS.                                          *
               00140 ;*    INPUT: HL=OPERAND 1 (H), OPERAND 2 (L)          *
               00150 ;*    OUTPUT:HL=OPERAND 1 XOR OPERAND 2               *
               00160 ;*****************************************************
               00170 ;
7F00 F5        00180 EXCLOR   PUSH    AF      ;SAVE REGISTERS
7F01 CD7F0A    00190          CALL    0A7FH   ;***GET OPERANDS***
```

```
7F04 7C         00200          LD      A,H        ;OPERAND 1
7F05 AD         00210          XOR     L          ;OPERAND 1 XOR OPERAND 2
7F06 6F         00220          LD      L,A        ;RESULT NOW IN L
7F07 2600       00230          LD      H,0        ;NOW IN HL
7F09 F1         00240          POP     AF         ;RESTORE REGISTER
7F0A C39A0A     00250          JP      0A9AH      ;***RETURN ARGUMENT***
7F0D C9         00260          RET                ;NON-BASIC RETURN
0000            00270          END
00000 TOTAL ERRORS
```

```
EXCLOR DECIMAL VALUES

 245, 205, 127, 10, 124, 173, 111, 38, 0, 241,
 195, 154, 10, 201

CHKSUM= 42
```

FILLME: FILL MEMORY

System Configuration

Model I, Model III, Model II Stand Alone.

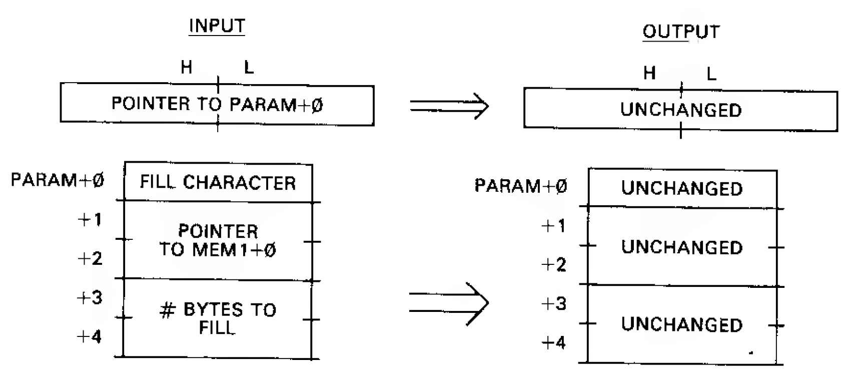
Description

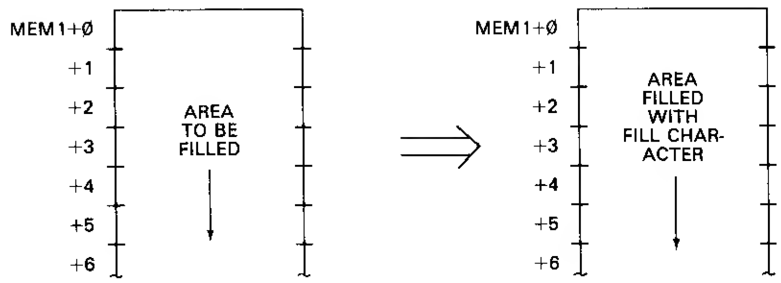
FILLME fills a block of memory with a given 8-bit value. Up to 65,535 bytes of memory can be filled.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first byte of the parameter block contains the fill value to be used. The next two bytes of the parameter block define the starting address for the block of memory to be filled in standard Z-80 address format, least significant byte followed by most significant byte. The next two bytes of the parameter block contain the number of bytes in the block to be filled.

On output, the block of memory has been filled; the parameter block remains unchanged.





Algorithm

The FILLME subroutine first picks up the number of bytes in the block and puts it into the BC register pair. Next, the starting address is put into the HL register pair. The A register is then loaded with the fill character.

The loop at FIL010 fills each byte in the memory block. The count in BC is decremented and the pointer in HL is adjusted to point to the next memory byte.

Sample Calling Sequence

```
NAME OF SUBROUTINE? FILLME
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0  1  65      "A" FILL CHARACTER
+ 1  2  50000   AREA TO FILL
+ 3  2  5       # OF BYTES
+ 5  0  0
MEMORY BLOCK 1 LOCATION? 50000
MEMORY BLOCK 1 VALUES?
+ 0  2  0  ┐
+ 2  2  0  │
+ 4  2  0  ├ INITIALIZE FILL AREA FOR EXAMPLE
+ 6  2  0  ┘
+ 8  0  0
MEMORY BLOCK 2 LOCATION?
MOVE SUBROUTINE TO? 38000
SUBROUTINE EXECUTED AT  38000
INPUT:              OUTPUT:
HL= 40000           HL= 40000
PARAM+ 0   65       PARAM+ 0   65
PARAM+ 1   80       PARAM+ 1   80
PARAM+ 2   195      PARAM+ 2   195
PARAM+ 3   5        PARAM+ 3   5
PARAM+ 4   0        PARAM+ 4   0
MEMB1+ 0   0        MEMB1+ 0   65 ┐
MEMB1+ 1   0        MEMB1+ 1   65 │
MEMB1+ 2   0        MEMB1+ 2   65 ├ FIVE "A"S FILLED
MEMB1+ 3   0        MEMB1+ 3   65 │
MEMB1+ 4   0        MEMB1+ 4   65 ┘
MEMB1+ 5   0        MEMB1+ 5   0
MEMB1+ 6   0        MEMB1+ 6   0
MEMB1+ 7   0        MEMB1+ 7   0

NAME OF SUBROUTINE?
```

Notes

1. The FILLME subroutine can be used to "zero" memory or to initialize the video display.

Program Listing

```
7F00                        ORG     7F00H           ;0520
             ;*****************************************************
             ;* FILL MEMORY. FILLS A BLOCK OF MEMORY WITH A GIVEN    *
             ;* VALUE.                                               *
             ;*     INPUT: HL=> PARAMETER BLOCK                      *
             ;*            PARAM+0=FILL CHARACTER                    *
             ;*            PARAM+1,+2=FILL STARTING ADDRESS          *
             ;*            PARAM+3,+4=# OF BYTES TO FILL, 1 TO 65535. *
             ;*                       0=65536                        *
             ;*     OUTPUT:BLOCK FILLED WITH GIVEN CHARACTER         *
             ;*****************************************************
             ;
7F00 F5      FILLME  PUSH    AF              ;SAVE REGISTERS
7F01 C5              PUSH    BC
7F02 D5              PUSH    DE
7F03 E5              PUSH    HL
7F04 DDE5            PUSH    IX
7F06 CD7F0A          CALL    0A7FH           ;***GET PB LOC'N***
7F09 E5              PUSH    HL              ;TRANSFER HL TO IX
7F0A DDE1            POP     IX
7F0C DD4604          LD      B,(IX+4)        ;PUT # OF BYTES IN BC
7F0F DD4E03          LD      C,(IX+3)
7F12 DD6602          LD      H,(IX+2)        ;PUT START IN HL
7F15 DD6E01          LD      L,(IX+1)
7F18 DD7E00          LD      A,(IX+0)        ;PUT FILL CHARACTER IN A
7F1B 77      FIL010  LD      (HL),A            ;FILL BYTE
7F1C 23              INC     HL                ;BUMP POINTER TO NEXT
7F1D 0B              DEC     BC                ;DECREMENT COUNT
7F1E 57              LD      D,A               ;SAVE A
7F1F 78              LD      A,B               ;TEST BC
7F20 B1              OR      C
7F21 7A              LD      A,D               ;RESTORE A
7F22 20F7            JR      NZ,FIL010         ;GO IF DONE
7F24 DDE1            POP     IX              ;RESTORE REGISTERS
7F26 E1              POP     HL
7F27 D1              POP     DE
7F28 C1              POP     BC
7F29 F1              POP     AF
7F2A C9              RET                     ;RETURN TO CALLING PROG
0000                 END
00000 TOTAL ERRORS
```

FILLME DECIMAL VALUES

```
245, 197, 213, 229, 221, 229, 205, 127, 10, 229,
221, 225, 221, 70, 4, 221, 78, 3, 221, 102,
2, 221, 110, 1, 221, 126, 0, 119, 35, 11,
87, 120, 177, 122, 32, 247, 221, 225, 225, 209,
193, 241, 201
```

CHKSUM= 17

FKBTST: FAST KEYBOARD TEST

System Configuration

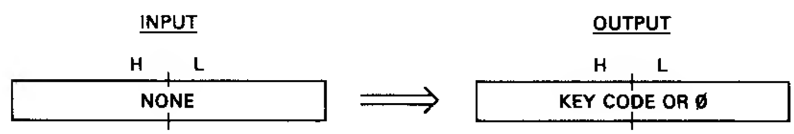
Model I, Model III.

Description

FKBTST is a "fast" keyboard test that tests for any key press and for five special keyboard keys, CLEAR, UP ARROW, DOWN ARROW, LEFT ARROW, and RIGHT ARROW. FKBTST returns a zero if no key is being pressed, a negative value if one of the special keys is being pressed, or a positive value if another key is being pressed. It can be used for games control or any other application where fast keyboard scanning is required.

Input/Output Parameters

No input parameters are required. On output, HL is returned with a zero for no keypress, −1 for CLEAR, −2 for UP ARROW, −3 for DOWN ARROW, −4 for LEFT ARROW, and −5 for RIGHT ARROW, or +1 through +127 for other key combinations.



Algorithm

The row address for the special keys is 3840H. This row is first read by an "LD A,(3840H)." The contents of A are then compared with the column bit configuration for the special keys (2, 8, 16, 32, and 64), and if there is a match the corresponding negative code is returned in HL. If there is no match, a "LD HL,(387FH)" is done. This reads all column bits into L. H is then cleared. If there was no key press, HL will now be set to zero.

Sample Calling Sequence

```
NAME OF SUBROUTINE? FKBTST
HL VALUE? Ø
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 45000
SUBROUTINE EXECUTED AT  45000
INPUT:              OUTPUT:
HL= Ø               HL= 65533   −3 = DOWN ARROW

NAME OF SUBROUTINE?
```

Notes

1. Detection of a special key will take about 60 microseconds, average time.

2. FKBTST may be used to detect multiple key presses, such as "JKL" or "123."

3. The SHIFT key is not tested.

Program Listing

```
7F00            00100          ORG     7F00H           ;0522
                00110 ;*****************************************************
                00120 ;* FAST KEYBOARD TEST. TESTS FOR ANY KEYPRESS AND FOR  *
                00130 ;* FIVE SPECIAL KEYS.                                  *
                00140 ;*     INPUT: NONE                                     *
                00150 ;*     OUTPUT:HL=0 FOR NO KEY PRESS,-1 FOR CLEAR,-2 FOR *
                00160 ;*            UP ARROW,-3 FOR DOWN ARROW,-4 FOR LEFT   *
                00170 ;*            ARROW, AND -5 FOR RIGHT ARROW, 1-127 FOR *
                00180 ;*            OTHER KEY COMBINATIONS.                  *
                00190 ;*****************************************************
                00200 ;
7F00 F5         00210 FKBTST   PUSH    AF              ;SAVE REGISTER
7F01 21FFFF     00220          LD      HL,-1           ;CLEAR CODE
7F04 3A4038     00230          LD      A,(3840H)       ;READ ROW
7F07 FE02       00240          CP      2               ;CLEAR?
7F09 2819       00250          JR      Z,FKB010        ;GO IF YES
7F0B 2B         00260          DEC     HL              ;UP ARROW CODE
7F0C FE08       00270          CP      8               ;UP ARROW?
7F0E 2814       00280          JR      Z,FKB010        ;GO IF YES
7F10 2B         00290          DEC     HL              ;DOWN ARROW CODE
7F11 FE10       00300          CP      16              ;DOWN ARROW?
7F13 280F       00310          JR      Z,FKB010        ;GO IF YES
7F15 2B         00320          DEC     HL              ;LEFT ARROW CODE
7F16 FE20       00330          CP      32              ;LEFT ARROW?
7F18 280A       00340          JR      Z,FKB010        ;GO IF YES
7F1A 2B         00350          DEC     HL              ;RIGHT ARROW CODE
7F1B FE40       00360          CP      64              ;RIGHT ARROW?
7F1D 2805       00370          JR      Z,FKB010        ;GO IF YES
7F1F 2A7F38     00380          LD      HL,(387FH)      ;READ ALL COLUMNS
7F22 2600       00390          LD      H,0             ;RESULT IN HL
7F24 F1         00400 FKB010   POP     AF              ;RESTORE REGISTER
7F25 C39A0A     00410          JP      0A9AH           ;***RETURN ARGUMENT***
7F28 C9         00420          RET                     ;NON-BASIC RETURN
0000            00430          END
00000 TOTAL ERRORS
```

```
FKBTST DECIMAL VALUES

245, 33, 255, 255, 58, 64, 56, 254, 2, 40,
25, 43, 254, 8, 40, 20, 43, 254, 16, 40,
15, 43, 254, 32, 40, 10, 43, 254, 64, 40,
5, 42, 127, 56, 38, 0, 241, 195, 154, 10,
201

CHKSUM= 29
```

FSETGR: FAST GRAPHICS SET/RESET

System Configuration

Model I, Model III.

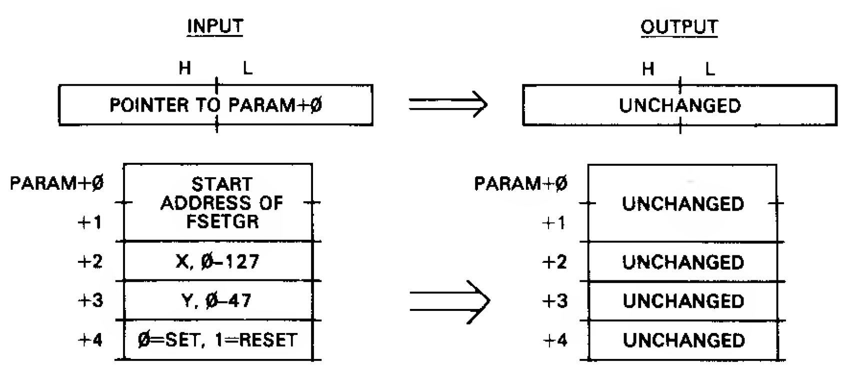
Description

FSETGR is a subroutine that sets or resets a given screen pixel. It is designed to perform screen actions rapidly and uses a table lookup structure to avoid the time-consuming processing present in other graphics subroutines. Any of the 6144 graphics pixels, arranged in 128 columns by 64 rows, may be set or reset. Previous to using FSETGR, the screen area to be utilized must have been cleared with graphics characters (80H).

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block are the starting address of the FSETGR subroutine, in standard Z-80 address format, least significant byte followed by most significant byte. The next byte of the parameter block is the x coordinate, 0 to 127. The next byte of the parameter block is the y coordinate, 0 to 47. The next byte of the parameter block is a set/reset flag. This byte is 0 if the pixel is to be set, or 0 if the pixel is to be reset.

On output, the pixel is set or reset, and the parameter block remains unchanged.



Algorithm

The FSETGR subroutine uses a table of 48 entries to implement fast graphics. Each entry in the table corresponds to one of the 48 rows of graphics and gives the actual memory address that contains the pixel and the mask to be used in processing the pixel. The first twelve bits of an entry represent the memory address when four zeroes are added to the twelve bits. The fifth entry of 3C44H, for example, represents 3C40H, the start of the fifth graphics row in memory. The last four bits represent the graphics mask to be used in processing, as we'll explain.

FSETGR first gets the y value from the parameter block. This y value is multiplied by 2 and added to the base address of FSETGR and TABLEA displacement; the result points to the TABLEA entry. The entry address is put into HL and IY. Next, the four least significant bits of HL are reset to mask out the graphics mask. HL now points to the start of the line containing the graphics byte.

Next, the x address is picked up from the parameter block. The x address is divided by two and added to the HL register. The HL register now points to the actual byte in memory containing the pixel to be processed.

Next, the A register is loaded with the least significant byte from the TABLEA table. This contains the graphics mask. The mask value is ANDed with 1FH to get only the mask. If X is even, the mask is left unchanged, as it represents the left-hand bit; if X is odd, the mask is shifted left for the right-hand bit.

The byte containing the pixel is now loaded into B. If a set is to be done, the mask in A is ORed with B and the result stored to set the pixel. If a reset is to be done, the complement of the mask in A is ANDed with B and the result stored to reset the pixel.

Sample Calling Sequence

```
NAME OF SUBROUTINE? FSETGR
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0  2  37000  START OF FSETGR
+ 2  1  64    ]
+ 3  1  24    ] X, Y = 64, 24
+ 4  1  0      SET
+ 5  0  0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 37000
SUBROUTINE EXECUTED AT  37000
INPUT:             OUTPUT:
HL= 40000          HL= 40000
PARAM+ 0   136     PARAM+ 0   136 ]
PARAM+ 1   144     PARAM+ 1   144 ]
PARAM+ 2   64      PARAM+ 2   64  ] UNCHANGED
PARAM+ 3   24      PARAM+ 3   24  ]
PARAM+ 4   0       PARAM+ 4   0   ]

NAME OF SUBROUTINE?
```

Notes

1. This subroutine can set/reset about 4000 points per second.

Program Listing

```
7F00                     ORG     7F00H           ;0522
                ;*******************************************************
                ;* FAST GRAPHICS SET/RESET. SETS/RESETS A GIVEN PIXEL.  *
                ;*    INPUT :HL=> PARAMETER BLOCK                       *
                ;*              PARAM+0,+1=START ADDRESS OF FSETGR      *
                ;*              PARAM+2=X, 0 TO 127                     *
                ;*              PARAM+3=Y, 0 TO 47                      *
                ;*              PARAM+4=SET/RESET FLAG. 0=SET, 1=RESET  *
                ;*    OUTPUT:PIXEL SET OR RESET                         *
                ;*******************************************************
                ;
7F00 F5         FSETGR  PUSH    AF              ;SAVE REGISTERS
7F01 C5                 PUSH    BC
7F02 D5                 PUSH    DE
7F03 E5                 PUSH    HL
7F04 DDE5               PUSH    IX
7F06 FDE5               PUSH    IY
7F08 CD7F0A             CALL    0A7FH           ;***GET PB LOC'N***
7F0B E5                 PUSH    HL              ;TRANSFER TO IX
```

```
7F0C DDE1       00290           POP     IX
7F0E 1600       00300           LD      D,0             ;ZERO D
7F10 DD5E03     00310           LD      E,(IX+3)        ;Y TO DE
7F13 CB23       00320           SLA     E               ;2*Y FOR TABLE LOOKUP
7F15 DD6E00     00330           LD      L,(IX+0)        ;GET BASE ADDRESS
7F18 DD6601     00340           LD      H,(IX+1)
7F1B 19         00350           ADD     HL,DE           ;ADD 2*Y
7F1C 015700     00360           LD      BC,TABLEA       ;TABLE DISPLACEMENT
7F1F 09         00370           ADD     HL,BC           ;POINT TO TABLE START
7F20 E5         00380           PUSH    HL              ;TRANSFER TO IY
7F21 FDE1       00390           POP     IY
7F23 FD7E00     00400           LD      A,(IY+0)        ;GET LINE START
7F26 E6E0       00410           AND     0E0H            ;MASK OUT MASK!
7F28 6F         00420           LD      L,A             ;LS BYTE NOW IN L
7F29 FD6601     00430           LD      H,(IY+1)
7F2C DD5E02     00440           LD      E,(IX+2)        ;GET X
7F2F 1600       00450           LD      D,0             ;NOW IN DE
7F31 CB3B       00460           SRL     E               ;NOW X/2
7F33 19         00470           ADD     HL,DE           ;POINT TO GRAPHICS BYTE
7F34 FD7E00     00480           LD      A,(IY+0)        ;GET BIT
7F37 E61F       00490           AND     1FH             ;GET MASK VALUE
7F39 DDCB0246   00500           BIT     0,(IX+2)        ;TEST LSB OF X FOR ODD/EVEN
7F3D 2802       00510           JR      Z,FSE020        ;GO IF LEFT
7F3F CB27       00520           SLA     A               ;RIGHT COLUMN
7F41 46         00530 FSE020    LD      B,(HL)          ;GET GRAPHICS BYTE
7F42 DDCB0446   00540           BIT     0,(IX+4)        ;TEST SET/RESET
7F46 2804       00550           JR      Z,FSE030        ;GO IF SET
7F48 2F         00560           CPL                     ;INVERT MASK
7F49 A0         00570           AND     B               ;RESET BIT
7F4A 1801       00580           JR      FSE040          ;CONTINUE
7F4C B0         00590 FSE030    OR      B               ;SET BIT
7F4D 77         00600 FSE040    LD      (HL),A          ;STORE GRAPHICS BYTE
7F4E FDE1       00610           POP     IY              ;RESTORE REGISTERS
7F50 DDE1       00620           POP     IX
7F52 E1         00630           POP     HL
7F53 D1         00640           POP     DE
7F54 C1         00650           POP     BC
7F55 F1         00660           POP     AF
7F56 C9         00670           RET                     ;RETURN TO CALLING PROG
0057            00680 TABLEA    EQU     $-FSETGR        ;DISP OF TABLE FROM START
7F57 013C       00690           DEFW    3C00H+1
7F59 043C       00700           DEFW    3C00H+4
7F5B 103C       00710           DEFW    3C00H+16
7F5D 413C       00720           DEFW    3C40H+1
7F5F 443C       00730           DEFW    3C40H+4
7F61 503C       00740           DEFW    3C40H+16
7F63 813C       00750           DEFW    3C80H+1
7F65 843C       00760           DEFW    3C80H+4
7F67 903C       00770           DEFW    3C80H+16
7F69 C13C       00780           DEFW    3CC0H+1
7F6B C43C       00790           DEFW    3CC0H+4
7F6D D03C       00800           DEFW    3CC0H+16
7F6F 013D       00810           DEFW    3D00H+1
7F71 043D       00820           DEFW    3D00H+4
7F73 103D       00830           DEFW    3D00H+16
7F75 413D       00840           DEFW    3D40H+1
7F77 443D       00850           DEFW    3D40H+4
7F79 503D       00860           DEFW    3D40H+16
7F7B 813D       00870           DEFW    3D80H+1
7F7D 843D       00880           DEFW    3D80H+4
7F7F 903D       00890           DEFW    3D80H+16
7F81 C13D       00900           DEFW    3DC0H+1
7F83 C43D       00910           DEFW    3DC0H+4
7F85 D03D       00920           DEFW    3DC0H+16
7F87 013E       00930           DEFW    3E00H+1
7F89 043E       00940           DEFW    3E00H+4
7F8B 103E       00950           DEFW    3E00H+16
7F8D 413E       00960           DEFW    3E40H+1
```

```
7F8F 443E       00970      DEFW    3E40H+4
7F91 503E       00980      DEFW    3E40H+16
7F93 813E       00990      DEFW    3E80H+1
7F95 843E       01000      DEFW    3E80H+4
7F97 903E       01010      DEFW    3E80H+16
7F99 C13E       01020      DEFW    3EC0H+1
7F9B C43E       01030      DEFW    3EC0H+4
7F9D D03E       01040      DEFW    3EC0H+16
7F9F 013F       01050      DEFW    3F00H+1
7FA1 043F       01060      DEFW    3F00H+4
7FA3 103F       01070      DEFW    3F00H+16
7FA5 413F       01080      DEFW    3F40H+1
7FA7 443F       01090      DEFW    3F40H+4
7FA9 503F       01100      DEFW    3F40H+16
7FAB 813F       01110      DEFW    3F80H+1
7FAD 843F       01120      DEFW    3F80H+4
7FAF 903F       01130      DEFW    3F80H+16
7FB1 C13F       01140      DEFW    3FC0H+1
7FB3 C43F       01150      DEFW    3FC0H+4
7FB5 D03F       01160      DEFW    3FC0H+16
0000            01170      END
00000 TOTAL ERRORS
```

```
FSETGR DECIMAL VALUES


245, 197, 213, 229, 221, 229, 253, 229, 205, 127,
10, 229, 221, 225, 22, 0, 221, 94, 3, 203,
35, 221, 110, 0, 221, 102, 1, 25, 1, 87,
0, 9, 229, 253, 225, 253, 126, 0, 230, 224,
111, 253, 102, 1, 221, 94, 2, 22, 0, 203,
59, 25, 253, 126, 0, 230, 31, 221, 203, 2,
70, 40, 2, 203, 39, 70, 221, 203, 4, 70,
40, 4, 47, 160, 24, 1, 176, 119, 253, 225,
221, 225, 225, 209, 193, 241, 201, 1, 60, 4,
60, 16, 60, 65, 60, 68, 60, 80, 60, 129,
60, 132, 60, 144, 60, 193, 60, 196, 60, 208,
60, 1, 61, 4, 61, 16, 61, 65, 61, 68,
61, 80, 61, 129, 61, 132, 61, 144, 61, 193,
61, 196, 61, 208, 61, 1, 62, 4, 62, 16,
62, 65, 62, 68, 62, 80, 62, 129, 62, 132,
62, 144, 62, 193, 62, 196, 62, 208, 62, 1,
63, 4, 63, 16, 63, 65, 63, 68, 63, 80,
63, 129, 63, 132, 63, 144, 63, 193, 63, 196,
63, 208, 63

CHKSUM= 69
```

INBLCK: INSERT BLOCK

System Configuration

Model I, Model III, Model II Stand Alone.

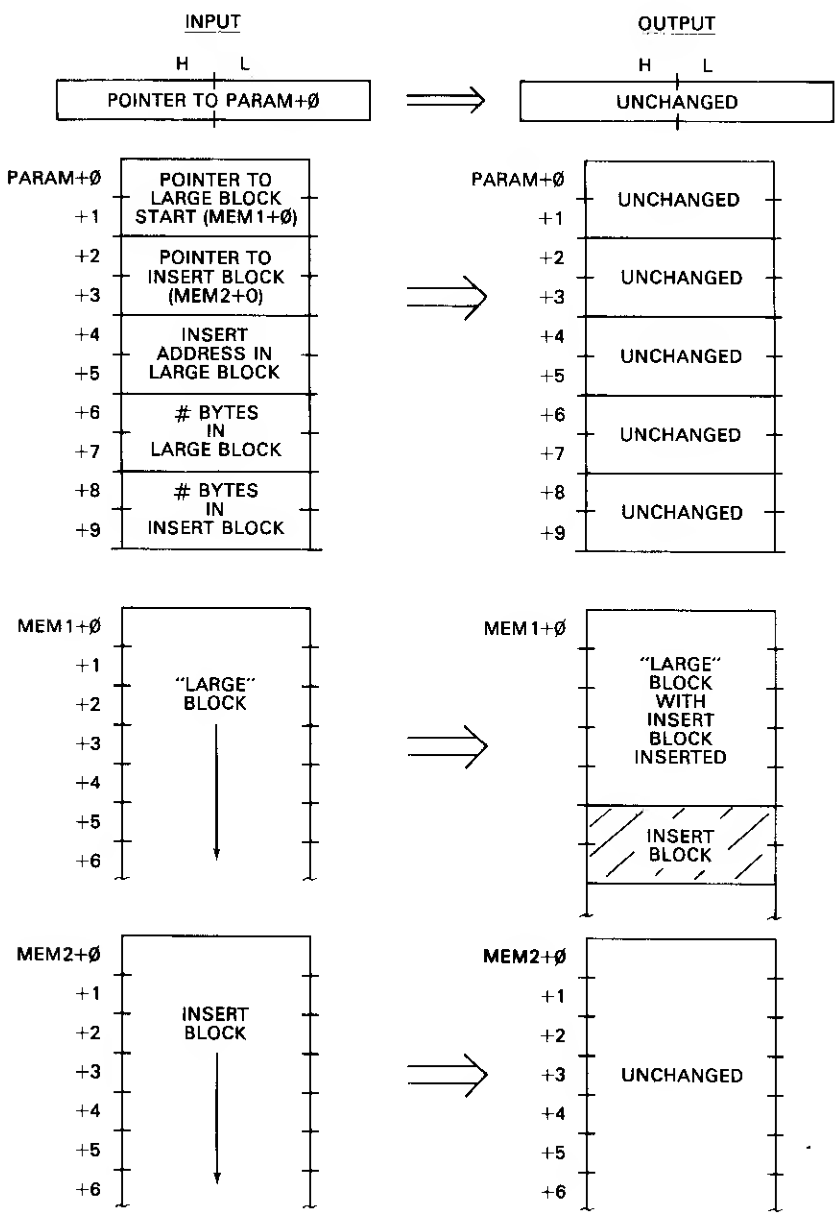
Description

INBLCK inserts a block in the middle of a larger block of memory. The block is inserted by moving down all bytes after the insertion point, as shown below. This subroutine could be used for inserting a block of text, for example, and moving the remaining text below the inserted block. Both the "larger block" and "insert block" may be any size, up to the limits of memory.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the address of the larger block in standard Z-80 address format, least significant byte followed by most significant byte. The next two bytes are the address of the insertion block in Z-80 address format. The next two bytes are the address of the insertion point in Z-80 address format. The next two bytes of the parameter block contain the number of bytes in the larger block; the next two bytes contain the number of bytes in the deletion block. Both are in standard Z-80 format.

On output, the contents of the parameter block remain unchanged. The insertion block has been inserted by a move of the insertion block into the insertion point.



Algorithm

The INBLCK subroutine performs the insertion by "opening up" space in the larger block for the bytes of the insertion block and then moving the insertion block into the space created.

Space is created by doing a block move downward of the area in the larger block from the insertion point to the end. This must be an LDDR to avoid replication of data. The LDDR is followed by an LDIR to insert the insertion block.

The LDDR must be set up with HL containing the address of the last byte of the larger block, DE containing the address of the last byte of the larger block plus the number of bytes in the insertion block, and BC containing the number of bytes in the larger block from the insertion point on. The HL address is found by adding the start of the larger block plus the number of bytes in the larger block minus one. This is saved in the stack for the LDDR. The BC count is found by subtracting the insert address from the end address and adding one. This is also saved for the LDDR. The DE address is found by adding the number of bytes in the insertion block to the end address. The move is then done by an LDDR.

The LDIR for the insert is then done after setting up DE with the address of the insertion point, HL with the address of the insertion block, and BC with the number of bytes of the insertion block.

Sample Calling Sequence

```
NAME OF SUBROUTINE? INBLCK
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0   2   50000  LARGE BLOCK START
+ 2   2   55000  INSERT BLOCK START
+ 4   2   50002  INSERT POINT
+ 6   2   5      5 BYTES IN LARGE BLOCK
+ 8   2   3      3 BYTES IN INSERT BLOCK
+ 10  0   0
MEMORY BLOCK 1 LOCATION? 50000
MEMORY BLOCK 1 VALUES?
+ 0   1   0  ┐                     ┐
+ 1   1   1  │                     │
+ 2   1   2  ├LARGE BLOCK          │
+ 3   1   3  │                     ├ INITIALIZE LARGE BLOCK FOR EXAMPLE
+ 4   1   4  ┘                     │
+ 5   1   5                        │
+ 6   1   6                        │
+ 7   1   7                        ┘
+ 8   1   0
+ 9   0   0
MEMORY BLOCK 2 LOCATION? 55000
MEMORY BLOCK 2 VALUES?
+ 0   1   255┐
+ 1   1   254├INSERT BLOCK
+ 2   1   253┘
+ 3   0   0
MOVE SUBROUTINE TO? 37000
SUBROUTINE EXECUTED AT  37000
INPUT:              OUTPUT:
HL= 40000           HL= 40000
```

PARAM+ 0	80	PARAM+ 0	80	UNCHANGED
PARAM+ 1	195	PARAM+ 1	195	
PARAM+ 2	216	PARAM+ 2	216	
PARAM+ 3	214	PARAM+ 3	214	
PARAM+ 4	82	PARAM+ 4	82	
PARAM+ 5	195	PARAM+ 5	195	
PARAM+ 6	5	PARAM+ 6	5	
PARAM+ 7	0	PARAM+ 7	0	
PARAM+ 8	3	PARAM+ 8	3	
PARAM+ 9	0	PARAM+ 9	0	
MEMB1+ 0	0	MEMB1+ 0	0	ORIGINAL DATA (8 BYTES OF NEW BLOCK)
MEMB1+ 1	1	MEMB1+ 1	1	
MEMB1+ 2	2	MEMB1+ 2	255	INSERTED DATA
MEMB1+ 3	3	MEMB1+ 3	254	
MEMB1+ 4	4	MEMB1+ 4	253	
MEMB1+ 5	5	MEMB1+ 5	2	ORIGINAL DATA
MEMB1+ 6	6	MEMB1+ 6	3	
MEMB1+ 7	7	MEMB1+ 7	4	
MEMB1+ 8	0	MEMB1+ 8	0	
MEMB2+ 0	255	MEMB2+ 0	255	UNCHANGED INSERT BLOCK
MEMB2+ 1	254	MEMB2+ 1	254	
MEMB2+ 2	253	MEMB2+ 2	253	

NAME OF SUBROUTINE?

Notes

1. The maximum number of bytes in either block may be 65,535.
2. The term "larger block" is somewhat misleading. The larger block may be smaller than the insertion block!
3. The insertion point must be within the larger block.

Program Listing

```
7F00            00100         ORG     7F00H           ;0520
                00110 ;*****************************************************
                00120 ;* INSERT BLOCK. INSERTS BLOCK IN MIDDLE OF LARGER BLOCK*
                00130 ;*    INPUT: HL=>PARAMETER BLOCK                         *
                00140 ;*           PARAM+0,+1=START ADDRESS OF LARGER BLOCK    *
                00150 ;*           PARAM+2,+3=START ADDRESS OF INSERT BLOCK    *
                00160 ;*           PARAM+4,+5=INSERT ADDRESS IN LARGER BLOCK   *
                00170 ;*           PARAM+6,+7=# OF BYTES IN LARGER BLOCK       *
                00180 ;*           PARAM+8,+9=# OF BYTES IN INSERT BLOCK       *
                00190 ;*    OUTPUT:INSERT BLOCK INSERTED IN LARGER BLOCK AND   *
                00200 ;*           FOLLOWING BYTES MOVED DOWN                  *
                00210 ;*****************************************************
                00220 ;
7F00 F5         00230 INBLCK  PUSH    AF              ;SAVE REGISTERS
7F01 C5         00240         PUSH    BC
7F02 D5         00250         PUSH    DE
7F03 E5         00260         PUSH    HL
7F04 DDE5       00270         PUSH    IX
7F06 CD7F0A     00280         CALL    0A7FH           ;***GET PB ADDRESS***
7F09 E5         00290         PUSH    HL              ;TRANSFER TO IX
7F0A DDE1       00300         POP     IX
7F0C DD6E00     00310         LD      L,(IX+0)        ;START OF LARGE BLOCK
7F0F DD6601     00320         LD      H,(IX+1)
7F12 DD4E06     00330         LD      C,(IX+6)        ;# OF BYTES IN LARGE BLK
7F15 DD4607     00340         LD      B,(IX+7)
7F18 09         00350         ADD     HL,BC           ;END OF LARGE BLK+1
7F19 2B         00360         DEC     HL
7F1A E5         00370         PUSH    HL              ;SAVE
```

```
7F1B DD4E04     00380          LD      C,(IX+4)       ;INSERT ADDRESS
7F1E DD4605     00390          LD      B,(IX+5)
7F21 B7         00400          OR      A              ;CLEAR CARRY
7F22 ED42       00410          SBC     HL,BC          ;FIND # TO MOVE
7F24 23         00420          INC     HL
7F25 D1         00430          POP     DE             ;SOURCE ADDRESS
7F26 E5         00440          PUSH    HL             ;SAVE # TO MOVE
7F27 DD6E08     00450          LD      L,(IX+8)       ;# OF BYTES IN INSERT BLK
7F2A DD6609     00460          LD      H,(IX+9)
7F2D 19         00470          ADD     HL,DE          ;FIND DESTINATION
7F2E EB         00480          EX      DE,HL          ;PUT IN PROPER REGISTERS
7F2F C1         00490          POP     BC             ;RESTORE #
7F30 EDB8       00500          LDDR                     ;MOVE BYTES
7F32 DD5E04     00510          LD      E,(IX+4)       ;INSERT ADDRESS
7F35 DD5605     00520          LD      D,(IX+5)
7F38 DD6E02     00530          LD      L,(IX+2)       ;SOURCE ADDRESS
7F3B DD6603     00540          LD      H,(IX+3)
7F3E DD4E08     00550          LD      C,(IX+8)       ;# OF BYTES TO MOVE
7F41 DD4609     00560          LD      B,(IX+9)
7F44 EDB0       00570          LDIR                     ;MOVE INSERT BLK TO INS PT
7F46 DDE1       00580          POP     IX             ;RESTORE REGISTERS
7F48 E1         00590          POP     HL
7F49 D1         00600          POP     DE
7F4A C1         00610          POP     BC
7F4B F1         00620          POP     AF
7F4C C9         00630          RET                    ;RETURN TO CALLING PROG
0000            00640          END
00000 TOTAL ERRORS
```

INBLCK DECIMAL VALUES

```
245, 197, 213, 229, 221, 229, 205, 127, 10, 229,
221, 225, 221, 110, 0, 221, 102, 1, 221, 78,
6, 221, 70, 7, 9, 43, 229, 221, 78, 4,
221, 70, 5, 183, 237, 66, 35, 209, 229, 221,
110, 8, 221, 102, 9, 25, 235, 193, 237, 184,
221, 94, 4, 221, 86, 5, 221, 110, 2, 221,
102, 3, 221, 78, 8, 221, 70, 9, 237, 176,
221, 225, 225, 209, 193, 241, 201
```

CHKSUM= 66

METEST: MEMORY TEST

System Configuration

Model I, Model III, Model II Stand Alone.

Description

This subroutine tests a given block of memory by a "PUSH/POP" method. One pass is made through the test with each byte of the block being tested twice, except for the starting and ending addresses of the block, which are tested only once. Pseudo-random data is used to test all locations.

The memory test is considered successful if pseudo-random data can be written into every location and then retrieved successfully. If data is retrieved and it is not identical to the pattern stored, the test immediately returns with an error flag set, a record of the failing location, the proper test pattern, and the erroneous result.

METEST should be called repetitively to exercise and test memory; the more iterations performed, the greater the confidence that memory is working.

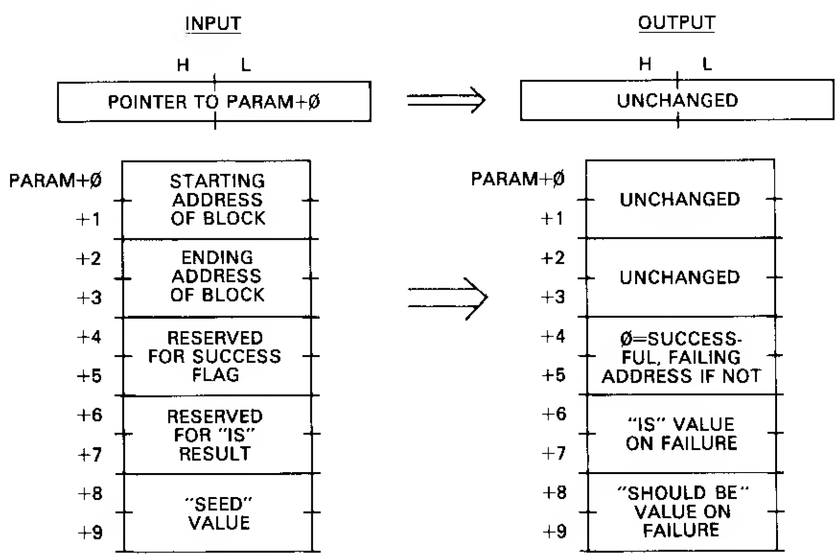
Input/Output Parameters

On input, the HL register pair points to a parameter block on entry to METEST. The first two bytes of the parameter block contain the starting address of the block to be tested. The next two bytes contain the ending address of the block. The ending address must be at least one location greater than the starting address.

The next four bytes are reserved for the test results.

The last two bytes contain a "seed" value for the memory test data. This seed value must be nonzero.

On output, PARAM+4, +5 contain the address of the failing location or the address of the failing location minus one if the test failed at any point. It contains a zero if the test was a success. PARAM+6, +7 and PARAM+8, +9 contain additional failure parameters.



The byte of PARAM+6 is the byte at the location equal to the failing address; the byte at PARAM+7 is the byte at a location one less than the failing address. Here's an example: If the failing word location is 20H, 80H (location 8020H) and PARAM+6, +7 contain a 63H, 32H with PARAM+8, +9 containing 67H, 32H, then the failing location is bit 2 of 8021H. If the failing word location is 8020H, PARAM+6, +7 contains a 66H, 32H and PARAM+8, +9 contains

67H, 33H then the failing location is bit 0 of 8020H. It is possible, of course, for both bytes to fail in the test.

A typical memory test first stores all zeroes into memory and then reads back the locations expecting to find all zeroes. It then stores all ones and reads back the data expecting all ones. At this point random data is usually stored and read back. METEST bypasses the first two tests of zeroes and ones.

More comprehensive memory tests are geared to the physical implementation of the type of memory. Various memory types have "worst case" test patterns. The dynamic memory used in the TRS-80s typically fails when adjacent locations are accessed. This test is an attempt to rapidly access adjacent locations by using stack instructions. Each PUSH or POP accesses two adjacent locations. Pseudo-random (repeatable) data is used for the test.

The pseudo-random data is generated from the last value in PARAM+8, +9. This value is multiplied by an odd power of 5, 125. The result is used as a test pattern for the two-byte PUSH and as the basis for the next generation of random data. The starting "seed" value can be maintained in later tests or varied to generate a new set of pseudo-random numbers.

Sample Calling Sequence

```
NAME OF SUBROUTINE? METEST
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0   2   42000  START ADDRESS
+ 2   2   48000  END ADDRESS
+ 4   2   0
+ 6   2   0
+ 8   2   1234   SEED VALUE
+ 10  0   0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 37800
SUBROUTINE EXECUTED AT  37800
INPUT:              OUTPUT:
HL= 40000           HL= 40000
PARAM+ 0   16       PARAM+ 0   16   ┐
PARAM+ 1   164      PARAM+ 1   164  ├UNCHANGED
PARAM+ 2   128      PARAM+ 2   128  │
PARAM+ 3   187      PARAM+ 3   187  ┘
PARAM+ 4   0        PARAM+ 4   0    ┐
PARAM+ 5   0        PARAM+ 5   0    ┴SUCCESS FLAG
PARAM+ 6   0        PARAM+ 6   82   ┐
PARAM+ 7   0        PARAM+ 7   238  ┴LAST "IS" VALUE
PARAM+ 8   210      PARAM+ 8   82   ┐
PARAM+ 9   4        PARAM+ 9   238  ┴LAST "SHOULD BE" VALUE

NAME OF SUBROUTINE?
```

Notes

1. Make certain ending location is at least one more than starting location.
2. Odd seed values generate a string of odd test values, even-seed values generate even test values.

```
7F00            00100           ORG     7F00H           ;0520
                00110 ;*****************************************************
                00120 ;* MEMORY TEST. TESTS A BLOCK OF MEMORY.             *
                00130 ;*    INPUT: HL=> PARAMETER BLOCK                    *
                00140 ;*           PARAM+0,+1=STARTING ADDRESS OF BLOCK    *
                00150 ;*           PARAM+2,+3=ENDING ADDRESS OF BLOCK      *
                00160 ;*           PARAM+4,+5 RESERVED FOR SUCCESS FLAG    *
                00170 ;*           PARAM+6,+7=RESERVED FOR "IS" RESULT     *
                00180 ;*           PARAM+8,+9=NON-ZERO "SEED" VALUE        *
                00190 ;*    OUTPUT:PARAM+4,+5=0 IF TEST SUCCESSFUL, FAILING *
                00200 ;*                  LOCATION IF TEST NOT SUCCESSFUL  *
                00210 ;*           PARAM+6,+7=TWO BYTES FROM MEMORY - "IS" *
                00220 ;*           PARAM+8,+9=TEST PATTERN - "S/B"         *
                00230 ;*****************************************************
                00240 ;
7F00 F5         00250 METEST    PUSH    AF              ;SAVE REGISTERS
7F01 C5         00260           PUSH    BC
7F02 D5         00270           PUSH    DE
7F03 E5         00280           PUSH    HL
7F04 DDE5       00290           PUSH    IX
7F06 FDE5       00300           PUSH    IY
7F08 CD7F0A     00310           CALL    0A7FH           ;***GET PB LOC'N***
7F0B E5         00320           PUSH    HL              ;TRANSFER TO IX
7F0C DDE1       00330           POP     IX
7F0E F3         00340           DI                      ;DISABLE INT FOR STACK
7F0F DD4E02     00350           LD      C,(IX+2)        ;END ADDRESS TO BC
7F12 DD4603     00360           LD      B,(IX+3)
7F15 FD210000   00370           LD      IY,0            ;ZERO IY FOR ADD SP
7F19 FD39       00380           ADD     IY,SP           ;TRANSFER CURNT SP TO IY
7F1B DD6E00     00390           LD      L,(IX+0)        ;GET START
7F1E DD6601     00400           LD      H,(IX+1)
7F21 DD7504     00410           LD      (IX+4),L        ;INITIALIZE CURRENT
7F24 DD7405     00420           LD      (IX+5),H
7F27 DD6E04     00430 MET010    LD      L,(IX+4)          ;CURRENT ADDRESS TO HL
7F2A DD6605     00440           LD      H,(IX+5)
7F2D 23         00450           INC     HL                ;BUMP CURRENT ADDRESS
7F2E DD7504     00460           LD      (IX+4),L          ;CURNT FOR FAILING LOC
7F31 DD7405     00470           LD      (IX+5),H
7F34 23         00480           INC     HL                ;1ST STACK ACTION AT -1
7F35 F9         00490           LD      SP,HL             ;SET SP FOR TEST
7F36 DD6E08     00500           LD      L,(IX+8)          ;GET SEED
7F39 DD6609     00510           LD      H,(IX+9)
7F3C 5D         00520           LD      E,L               ;PUT IN HL AND DE
7F3D 54         00530           LD      D,H
7F3E 3E07       00540           LD      A,7               ;LOOP COUNT FOR SHIFT
7F40 29         00550 MET020    ADD     HL,HL             ;SEED*2
7F41 3D         00560           DEC     A                 ;DECREMENT LOOP COUNT
7F42 20FC       00570           JR      NZ,MET020         ;7 TIMES=TIMES 128
7F44 B7         00580           OR      A
7F45 ED52       00590           SBC     HL,DE             ;TIMES 127
7F47 B7         00600           OR      A
7F48 ED52       00610           SBC     HL,DE             ;TIMES 126
7F4A B7         00620           OR      A
7F4B ED52       00630           SBC     HL,DE             ;TIMES 125
7F4D DD7508     00640           LD      (IX+8),L          ;STORE NEW SEED
7F50 DD7409     00650           LD      (IX+9),H
7F53 E5         00660           PUSH    HL                ;ACTUAL TEST HERE
7F54 D1         00670           POP     DE                ;PUSH AND RETRIEVE
7F55 B7         00680           OR      A                 ;CLEAR CARRY
7F56 ED52       00690           SBC     HL,DE             ;TEST FOR EQUAL
7F58 19         00700           ADD     HL,DE             ;RESTORE "IS"
7F59 DD7506     00710           LD      (IX+6),L          ;SAVE IN "IS"
7F5C DD7407     00720           LD      (IX+7),H
```

```
7F5F 2012      00730          JR      NZ,MET030       ;GO IF NOT EQUAL
7F61 DD6E04    00740          LD      L,(IX+4)        ;GET CURRENT LOCATION
7F64 DD6605    00750          LD      H,(IX+5)
7F67 B7        00760          OR      A               ;CLEAR CARRY
7F68 ED42      00770          SBC     HL,BC           ;TEST FOR END
7F6A 20BB      00780          JR      NZ,MET010       ;LOOP FOR NXT TST OF 2
7F6C AF        00790          XOR     A               ;TEST SUCCESSFUL HERE
7F6D DD7704    00800          LD      (IX+4),A        ;SET SUCCESSFUL FLAG
7F70 DD7705    00810          LD      (IX+5),A
7F73 FDF9      00820 MET030   LD      SP,IY           ;RESTORE SP
7F75 FDE1      00830          POP     IY              ;RESTORE REGISTERS
7F77 DDE1      00840          POP     IX
7F79 E1        00850          POP     HL
7F7A D1        00860          POP     DE
7F7B C1        00870          POP     BC
7F7C F1        00880          POP     AF
7F7D C9        00890          RET                     ;RETURN TO CALLING PROG
0000           00900          END
00000 TOTAL ERRORS
```

METEST DECIMAL VALUES

```
245, 197, 213, 229, 221, 229, 253, 229, 205, 127,
10, 229, 221, 225, 243, 221, 78, 2, 221, 70,
3, 253, 33, 0, 0, 253, 57, 221, 110, 0,
221, 102, 1, 221, 117, 4, 221, 116, 5, 221,
110, 4, 221, 102, 5, 35, 221, 117, 4, 221,
116, 5, 35, 249, 221, 110, 8, 221, 102, 9,
93, 84, 62, 7, 41, 61, 32, 252, 183, 237,
82, 183, 237, 82, 183, 237, 82, 221, 117, 8,
221, 116, 9, 229, 209, 183, 237, 82, 25, 221,
117, 6, 221, 116, 7, 32, 18, 221, 110, 4,
221, 102, 5, 183, 237, 66, 32, 187, 175, 221,
119, 4, 221, 119, 5, 253, 249, 253, 225, 221,
225, 225, 209, 193, 241, 201
```

CHKSUM= 51

MLEBYE: FAST 8 BY 8 MULTIPLY

System Configuration

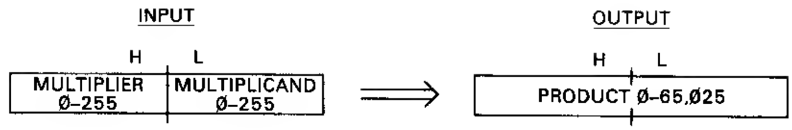
Model I, Model III, Model II Stand Alone.

Description

MLEBYE multiplies an 8-bit binary number by an 8-bit binary number to give a 16-bit product. The multiply is a "fast" multiply that operates twice as fast as conventional multiplies. The multiply is an "unsigned" multiply, where both operands are treated as 8-bit absolute numbers.

Input/Output Parameters

On input, the H register contains the 8-bit multiplier and the L register contains the 8-bit multiplicand. On output, HL contains the 16-bit product.



Algorithm

The MLEBYE subroutine performs the multiply by a bit-by-bit multiply in eight steps. To reduce overhead, "straight-line" coding rather than a loop structure is used.

The multiplicand is put into BC and the multiplier into H. The L register is cleared. The HL register is used to shift out multiplier bits from the left end into the carry and to hold the partial product in the L register end. The HL register is shifted left eight times. For each shift, a multipler bit from H is tested. If it is a one bit, the multiplicand in C is added to HL by an "ADD HL, BC"; if it is a zero, nothing is done. The next shift moves the partial product in L toward the left. At the end of the eight steps, the entire multiplier has been shifted out of H, and HL holds the 16-bit product.

Sample Calling Sequence

```
NAME OF SUBROUTINE? MLEBYE
HL VALUE? 65535   MULTIPLIER = 255, MULTIPLICAND = 255
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 55000
SUBROUTINE EXECUTED AT  55000
INPUT:             OUTPUT:
HL= 65535          HL= 65025   RESULT = 255 x 255

NAME OF SUBROUTINE?
```

Notes

1. Maximum multiplier is 255. Maximum multiplicand is 255. The maximum product will be 65,535.

Program Listing

```
7F00                    ORG     7F00H           ;0520
                ;**********************************************************
                ;* FAST 8 BIT BY 8 BIT MULTIPLY TO YIELD 16 BIT PRODUCT.*
                ;*    INPUT: HL=MULTIPLIER IN H, MULTIPLICAND IN L      *
                ;*    OUTPUT:HL=16-BIT PRODUCT, 0-65535                 *
                ;**********************************************************
                ;
7F00 C5         MLEBYE  PUSH    BC              ;SAVE REGISTER
7F01 CD7F0A             CALL    0A7FH           ;***GET HL***
7F04 4D                 LD      C,L             ;MULTIPLICAND TO C
7F05 0600               LD      B,0             ;NOW IN BC
7F07 68                 LD      L,B             ;0 TO L
7F08 29                 ADD     HL,HL           ;SHIFT MULTIPLIER, PRODUCT
7F09 3001               JR      NC,MLE010       ;GO IF MULTIPLIER BIT=0
7F0B 09                 ADD     HL,BC           ;ADD MULTIPLICAND
7F0C 29         MLE010  ADD     HL,HL
```

```
7F0D 3001       00260           JR      NC,MLE020
7F0F 09         00270           ADD     HL,BC
7F10 29         00280 MLE020    ADD     HL,HL
7F11 3001       00290           JR      NC,MLE030
7F13 09         00300           ADD     HL,BC
7F14 29         00310 MLE030    ADD     HL,HL
7F15 3001       00320           JR      NC,MLE040
7F17 09         00330           ADD     HL,BC
7F18 29         00340 MLE040    ADD     HL,HL
7F19 3001       00350           JR      NC,MLE050
7F1B 09         00360           ADD     HL,BC
7F1C 29         00370 MLE050    ADD     HL,HL
7F1D 3001       00380           JR      NC,MLE060
7F1F 09         00390           ADD     HL,BC
7F20 29         00400 MLE060    ADD     HL,HL
7F21 3001       00410           JR      NC,MLE070
7F23 09         00420           ADD     HL,BC
7F24 29         00430 MLE070    ADD     HL,HL
7F25 3001       00440           JR      NC,MLE080
7F27 09         00450           ADD     HL,BC
7F28 C1         00460 MLE080    POP     BC                      ;RESTORE REGISTER
7F29 C39A0A     00470           JP      0A9AH                   ;***RETURN ARGUMENT***
7F2C C9         00480           RET                             ;NON-BASIC RETURN
0000            00490           END
00000 TOTAL ERRORS
```

```
MLEBYE DECIMAL VALUES


 197, 205, 127, 10, 77, 6, 0, 104, 41, 48,
 1, 9, 41, 48, 1, 9, 41, 48, 1, 9,
 41, 48, 1, 9, 41, 48, 1, 9, 41, 48,
 1, 9, 41, 48, 1, 9, 41, 48, 1, 9,
 193, 195, 154, 10, 201

CHKSUM= 223
```

MLSBYS: SIXTEEN BY SIXTEEN MULTIPLY

System Configuration

Model I, Model III, Model II Stand Alone.

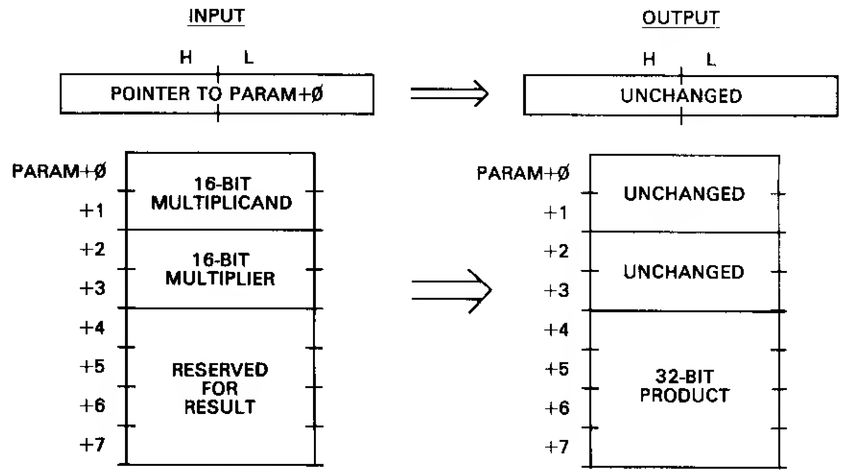
Description

MLSBYS multiplies a 16-bit binary number by a 16-bit binary number. The multiply is an "unsigned" multiply, where both numbers are considered to be absolute numbers without sign. A 32-bit product is returned.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the 16-bit multiplicand. The next two bytes of the parameter block contain a 16-bit multiplier. Both are in Z-80 16-bit format. The next four bytes of the parameter block are reserved for the 32-bit quotient.

On output, PARAM+3 to PARAM+6 hold the 32-bit product, arranged in next ms, ms, ls, next ls format. The contents of the remainder of the parameter block remain unchanged.



Algorithm

The MLSBYS subroutine performs the multiply by a "bit-by-bit" multiply in 16 iterations. The multiplier bits are tested from left to right. For each one bit in the multiplier, the multiplicand is added to a "partial product." The partial product is shifted left with each iteration. At the end of 16 iterations, all multiplier bits have been tested, and the partial product contains the true 32-bit product of the multiply.

The multiplicand is first put into BC, and the multiplier in DE. The A register is initialized with the iteration count of 16. The HL register is cleared to 0. The DE and HL registers will contain the partial product and will be shifted toward the left.

The code at MLS010 is the 16-iteration loop of MLSBYS. For each iteration, DE, HL is shifted one bit left. As it is shifted, the multiplier bit from DE goes into the carry. If the carry is set (multiplier bit is a one), the multiplicand in BC is added to the partial product. If the carry is reset (multiplier bit is a zero), no add is done. At the end of 16 iterations DE, HL contains the 32-bit product.

Sample Calling Sequence

```
NAME OF SUBROUTINE? MLSBYS
HL VALUE? 38888
PARAMETER BLOCK LOCATION? 38888
PARAMETER BLOCK VALUES?
+ 0  2  65535   MULTIPLICAND
+ 2  2  65535   MULTIPLIER
+ 4  2  0 ]
+ 6  2  0 |-INITIALIZE RESULT FOR EXAMPLE
+ 8  0  0 ]
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 40000
SUBROUTINE EXECUTED AT  40000
INPUT:           OUTPUT:
HL= 38888        HL= 38888
```

```
PARAM+ 0   255     PARAM+ 0   255  ┐
PARAM+ 1   255     PARAM+ 1   255  ├ UNCHANGED
PARAM+ 2   255     PARAM+ 2   255  │
PARAM+ 3   255     PARAM+ 3   255  ┘
PARAM+ 4   0       PARAM+ 4   254  ┐
PARAM+ 5   0       PARAM+ 5   255  ├ 254, 255, 1, 0 = 255, 254, 0,
PARAM+ 6   0       PARAM+ 6   1    │ 1 = 4, 294, 836, 225
PARAM+ 7   0       PARAM+ 7   0    ┘

NAME OF SUBROUTINE?
```

Notes

1. Maximum multiplier is 65,535. Maximum multiplicand is 65,535.
2. Note that the product is in 1,0,3,2 order.

Program Listing

```
7F00                       ORG     7F00H           ;0522
                   ;*****************************************************
                   ;* SIXTEEN BY SIXTEEN MULTIPLY TO YIELD 32-BIT PRODUCT. *
                   ;*     INPUT: HL=> PARAMETER BLOCK                        *
                   ;*            PARAM+0,+1=MULTIPLICAND                     *
                   ;*            PARAM+2,+3=MULTIPLIER                       *
                   ;*            PARAM+4,+5,+6,+7=RESERVED FOR PRODUCT       *
                   ;*     OUTPUT:PARA+4,+5,+6,+7 HOLD 32-BIT PRODUCT         *
                   ;*****************************************************
                   ;
7F00 F5            MLSBYS  PUSH    AF              ;SAVE REGISTERS
7F01 C5                    PUSH    BC
7F02 D5                    PUSH    DE
7F03 E5                    PUSH    HL
7F04 DDE5                  PUSH    IX
7F06 CD7F0A                CALL    0A7FH           ;***GET PB LOC'N***
7F09 E5                    PUSH    HL              ;TRANSFER TO IX
7F0A DDE1                  POP     IX
7F0C DD4E00                LD      C,(IX+0)        ;PUT MULTIPLICAND IN BC
7F0F DD4601                LD      B,(IX+1)
7F12 DD5E02                LD      E,(IX+2)        ;PUT MULTIPLIER IN DE
7F15 DD5603                LD      D,(IX+3)
7F18 3E10                  LD      A,16            ;ITERATION COUNT
7F1A 210000                LD      HL,0            ;ZERO PARTIAL PRODUCT
7F1D 29            MLS010  ADD     HL,HL             ;SHIFT PARTIAL PROD LEFT
7F1E EB                    EX      DE,HL             ;GET MS 16 BITS
7F1F ED6A                  ADC     HL,HL             ;SHIFT PART PROD PLUS C
7F21 EB                    EX      DE,HL             ;RESTORE UPPER 16 BITS
7F22 3004                  JR      NC,MLS020         ;GO IF MULTIPLIER BIT=0
7F24 09                    ADD     HL,BC             ;ADD IN MULTPLICAND
7F25 3001                  JR      NC,MLS020         ;GO IF NO CARRY
7F27 13                    INC     DE                ;BUMP UPPER 16 BITS
7F28 3D            MLS020  DEC     A                 ;DECREMENT ITERATION CNT
7F29 20F2                  JR      NZ,MLS010         ;LOOP FOR 16 ITERATIONS
7F2B DD7304                LD      (IX+4),E        ;STORE PRODUCT
7F2E DD7205                LD      (IX+5),D
7F31 DD7506                LD      (IX+6),L
7F34 DD7407                LD      (IX+7),H
7F37 DDE1                  POP     IX              ;RESTORE REGISTERS
7F39 E1                    POP     HL
7F3A D1                    POP     DE
7F3B C1                    POP     BC
7F3C F1                    POP     AF
7F3D C9                    RET                     ;RETURN TO CALLING PROG
0000                       END
00000 TOTAL ERRORS
```

MLSBYS DECIMAL VALUES

```
245, 197, 213, 229, 221, 229, 205, 127, 1Ø, 229,
221, 225, 221, 78, Ø, 221, 7Ø, 1, 221, 94,
2, 221, 86, 3, 62, 16, 33, Ø, Ø, 41,
235, 237, 1Ø6, 235, 48, 4, 9, 48, 1, 19,
61, 32, 242, 221, 115, 4, 221, 114, 5, 221,
117, 6, 221, 116, 7, 221, 225, 225, 2Ø9, 193,
241, 2Ø1
```

CHKSUM= 2Ø1

MOVEBL: MOVE BLOCK

System Configuration

Model I, Model III, Model II Stand Alone.

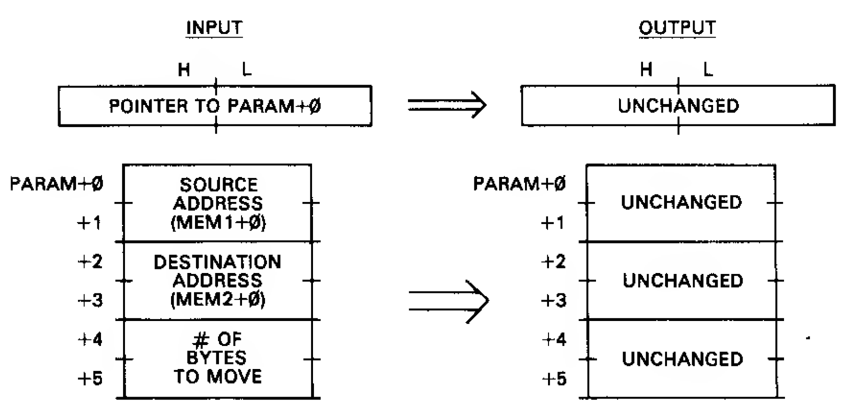
Description

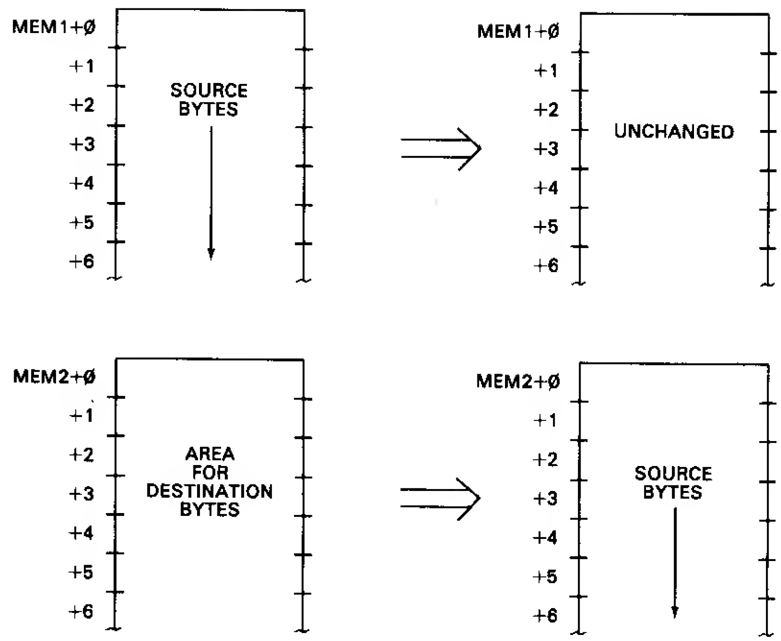
MOVBLK moves a block of memory to another block of memory. The blocks may be overlapping; a check is made for the proper direction of the move to prevent replication of data if the block move is made in the wrong direction. Any number of bytes up to the limit of memory may be moved.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the address of the source block in standard Z-80 address format, least significant byte followed by most significant byte. The next two bytes are the address of the destination block in Z-80 address format. The next two bytes of the parameter block contain the number of bytes to move in Z-80 format.

On output, the parameter block contents remain unchanged. The source block has been moved to the destination block area.





Algorithm

The main concern in MOVEBL is to test for either a "beginning to end" move or an "end to beginning" move. The wrong choice will replicate data in the block when the source and destination areas are overlapping. A test for overlap is not done, since it is simpler to choose either an LDIR or LDDR based on the relationship of the starting addresses.

The source address is put into HL, the destination address into DE, and the number of bytes into BC. A comparison is then done by subtracting the destination address from the source address. If the result is positive, the source address is less than the destination and an LDIR will perform the move with no conflict. If the result is negative, an LDDR must be done. In this case the source and destination addresses are recomputed so that they point to the end of the blocks for the LDDR.

Sample Calling Sequence

```
NAME OF SUBROUTINE? MOVEBL
HL VALUE? 45000
PARAMETER BLOCK LOCATION? 45000
PARAMETER BLOCK VALUES?
+ 0  2  50000  SOURCE ADDRESS
+ 2  2  50001  DESTINATION ADDRESS
+ 4  2  5      5 BYTES
+ 6  0  0
MEMORY BLOCK 1 LOCATION? 50000
MEMORY BLOCK 1 VALUES?
+ 0  1  0 ┐
+ 1  1  1 │
+ 2  1  2 │
+ 3  1  3 ├INITIALIZE SOURCE FOR EXAMPLE
+ 4  1  4 │
+ 5  1  5 │
+ 6  1  6 ┘
+ 7  0  0
```

```
MEMORY BLOCK 2 LOCATION?
MOVE SUBROUTINE TO? 37777
SUBROUTINE EXECUTED AT  37777
INPUT:              OUTPUT:
HL= 45000           HL= 45000
PARAM+ 0  80        PARAM+ 0  80
PARAM+ 1  195       PARAM+ 1  195
PARAM+ 2  81        PARAM+ 2  81     UNCHANGED
PARAM+ 3  195       PARAM+ 3  195
PARAM+ 4  5         PARAM+ 4  5
PARAM+ 5  0         PARAM+ 5  0
MEMB1+ 0  0         MEMB1+ 0  0
MEMB1+ 1  1         MEMB1+ 1  0
MEMB1+ 2  2         MEMB1+ 2  1
MEMB1+ 3  3         MEMB1+ 3  2      DESTINATION
MEMB1+ 4  4         MEMB1+ 4  3
MEMB1+ 5  5         MEMB1+ 5  4
MEMB1+ 6  6         MEMB1+ 6  6

NAME OF SUBROUTINE?
```

Notes

1. The number of bytes moved may be 1 to 65,536 (0 is 65,536).

Program Listing

```
7F00                    ORG     7F00H           ;0612
               ;*****************************************************
               ;* MOVE BLOCK. MOVES BLOCK OF DATA FROM SOURCE AREA TO  *
               ;* DESTINATION AREA. AREAS MAY BE OVERLAPPING.          *
               ;*     INPUT: HL=> PARAMETER BLOCK                      *
               ;*            PARAM+0,+1=SOURCE ADDRESS                 *
               ;*            PARAM+2,+3=DESTINATION ADDRESS            *
               ;*            PARAM+4,+5=# OF BYTES TO MOVE             *
               ;*     OUTPUT:BLOCK MOVED                               *
               ;*****************************************************
               ;
7F00 C5        MOVEBL   PUSH    BC              ;SAVE REGISTERS
7F01 D5                 PUSH    DE
7F02 E5                 PUSH    HL
7F03 DDE5               PUSH    IX
7F05 CD7F0A             CALL    0A7FH           ;***GET PB LOC'N***
7F08 E5                 PUSH    HL              ;TRANSFER TO IX
7F09 DDE1               POP     IX
7F0B DD6E00             LD      L,(IX+0)        ;PUT SOURCE ADDRESS IN HL
7F0E DD6601             LD      H,(IX+1)
7F11 DD5E02             LD      E,(IX+2)        ;PUT DESTINATION ADD IN DE
7F14 DD5603             LD      D,(IX+3)
7F17 DD4E04             LD      C,(IX+4)        ;PUT BYTE COUNT IN BC
7F1A DD4605             LD      B,(IX+5)
7F1D E5                 PUSH    HL              ;SAVE SOURCE ADDRESS
7F1E B7                 OR      A               ;CLEAR CARRY
7F1F ED52               SBC     HL,DE           ;COMPARE SOURCE TO DEST ADDR
7F21 CB7C               BIT     7,H             ;TEST SIGN
7F23 E1                 POP     HL              ;RESTORE SOURCE ADDRESS
7F24 2004               JR      NZ,MOV020       ;GO IF LDDR REQUIRED
7F26 EDB0               LDIR                    ;MOVE BLOCK
7F28 1808               JR      MOV030          ;GO TO CLEANUP
7F2A 0B        MOV020   DEC     BC              ;# OF BYTES-1
7F2B 09                 ADD     HL,BC           ;POINT TO NEW SOURCE
7F2C EB                 EX      DE,HL           ;GET DESTINATION
7F2D 09                 ADD     HL,BC           ;POINT TO NEW DESTINATION
7F2E EB                 EX      DE,HL           ;RESTORE
```

```
7F2F 03                  INC   BC         ;# BYTES
7F30 EDB8                LDDR               ;MOVE BLOCK
7F32 DDE1         MOV030 POP   IX         ;RESTORE REGISTERS
7F34 E1                  POP   HL
7F35 D1                  POP   DE
7F36 C1                  POP   BC
7F37 C9                  RET              ;RETURN TO CALLING PROGRAM
0000                     END
00000 TOTAL ERRORS
```

MOVEBL DECIMAL VALUES

```
197, 213, 229, 221, 229, 205, 127, 10, 229, 221,
225, 221, 110, 0, 221, 102, 1, 221, 94, 2,
221, 86, 3, 221, 78, 4, 221, 70, 5, 229,
183, 237, 82, 203, 124, 225, 32, 4, 237, 176,
24, 8, 11, 9, 235, 9, 235, 3, 237, 184,
221, 225, 225, 209, 193, 201
```

CHKSUM= 12

MPADDN: MULTIPLE-PRECISION ADD

System Configuration

Model I, Model III, Model II Stand Alone.

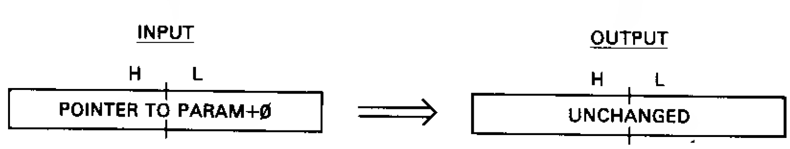
Description

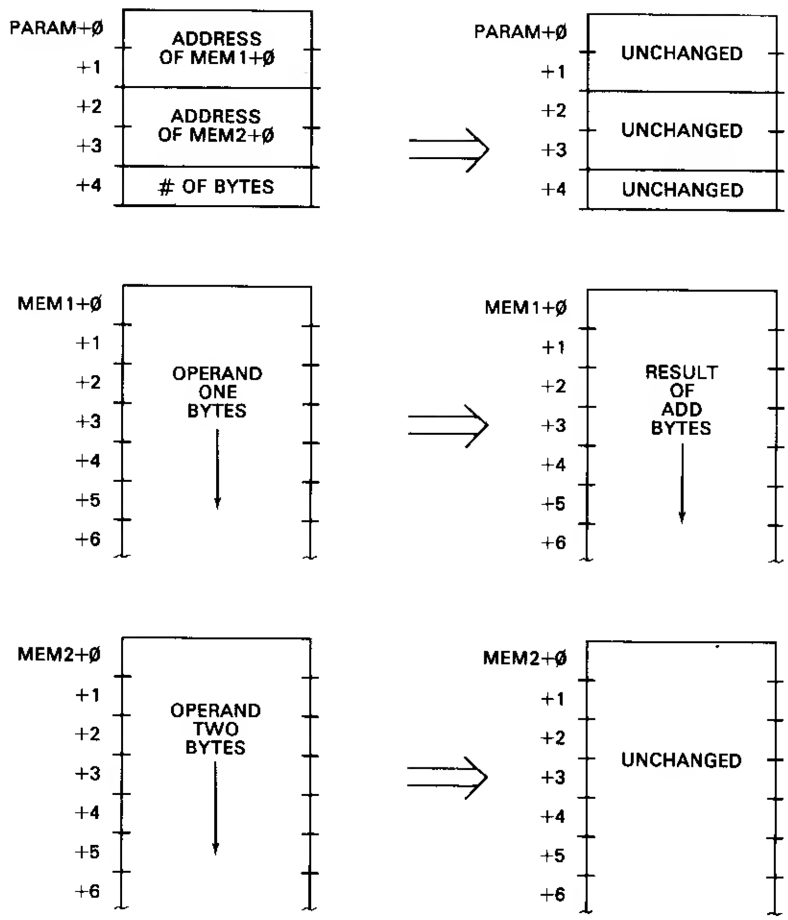
MPADDN adds a "source" string of bytes to a "destination" string of bytes and puts the result of the add into the destination string. Each of the two strings is a multiple-precision binary number. Each of the two strings is assumed to be the same length. The length of each string may be any number from 1 through 255 or 0, which is 256 bytes.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the address of the destination string in standard Z-80 address format, least significant byte followed by most significant byte. The next two bytes of the parameter block contain the address of the source string in the same format. The next byte of the parameter block contains the number of bytes in the two operands.

On output, the parameter block and source string are unchanged. The destination string contains the result of the multiple-precision add.





Algorithm

The MPADDN subroutine performs one add for each byte in the operands. The destination string address and source string address are first picked up from the parameter block and put into DE and HL, respectively. The number of bytes in the add is then picked up and put into the BC register pair. This number minus one is then added to the source and destination pointers so that they point to the least significant bytes of the source and destination strings. The number of bytes is then put into the B register for loop control.

The next destination byte is then picked up from the destination string (DE register pointer). An ADC is made of the two source string digits (HL register pointer). The result is then stored in the destination string.

The source and destination string pointers are then decremented by one to point to the next most significant two bytes of each operand. The B register count is then decremented by a DJNZ, and a loop back to MPA010 is made for the next add.

The carry is cleared before the first add, but successive adds add in the carry from the preceding operation. If the destination operand was 00H, F5H, 6EH, 11H and the source operand was 00H, FFH, 77H, 33H, then the number of operand bytes must be 4. The result in the destination operand would be 01H, F4H, E5H, 44H. Note that the result may be one bit larger than the original number of bits in the operands.

Sample Calling Sequence

```
NAME OF SUBROUTINE? MPADDN
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0  2  42000 POINTS TO DESTINATION
+ 2  2  44000 POINTS TO SOURCE
+ 4  2  5     5 BYTES
+ 6  0  0
MEMORY BLOCK 1 LOCATION? 42000
MEMORY BLOCK 1 VALUES?
+ 0  1  255 ┐
+ 1  1  255 │
+ 2  1  255 ├ DESTINATION = FFFFFFFEFFH
+ 3  1  254 │
+ 4  1  255 ┘
+ 5  0  0
MEMORY BLOCK 2 LOCATION? 44000
MEMORY BLOCK 2 VALUES?
+ 0  1  0   ┐
+ 1  1  0   │
+ 2  1  1   ├ SOURCE = 0000010001H
+ 3  1  0   │
+ 4  1  1   ┘
+ 5  0  0
MOVE SUBROUTINE TO? 38000
SUBROUTINE EXECUTED AT  38000
INPUT:              OUTPUT:
HL= 40000           HL= 40000
PARAM+ 0  16        PARAM+ 0  16  ┐
PARAM+ 1  164       PARAM+ 1  164 │
PARAM+ 2  224       PARAM+ 2  224 ├ UNCHANGED
PARAM+ 3  171       PARAM+ 3  171 │
PARAM+ 4  5         PARAM+ 4  5   │
PARAM+ 5  0         PARAM+ 5  0   ┘
MEMB1+ 0  255       MEMB1+ 0  0   ┐
MEMB1+ 1  255       MEMB1+ 1  0   │
MEMB1+ 2  255       MEMB1+ 2  0   ├ RESULT = 00000000FF00H
MEMB1+ 3  254       MEMB1+ 3  255 │
MEMB1+ 4  255       MEMB1+ 4  0   ┘
MEMB2+ 0  0         MEMB2+ 0  0   ┐
MEMB2+ 1  0         MEMB2+ 1  0   │
MEMB2+ 2  1         MEMB2+ 2  1   ├ UNCHANGED
MEMB2+ 3  0         MEMB2+ 3  0   │
MEMB2+ 4  1         MEMB2+ 4  1   ┘

NAME OF SUBROUTINE?
```

Notes

1. The destination string is fixed length. Leading zero bytes must precede the operands to handle the result, which may be one bit larger than either of the operands.

2. This may be either a "signed" or "unsigned" add. If a two's complement number is used, then the sign must be "sign extended" to the more significant bits of the operands.

```
7F00                     ORG     7F00H           ;0522
                ;***************************************************
                ;* MULTIPLE-PRECISION ADD. ADDS TWO MULTIPLE-PRECISION  *
                ;* OPERANDS, ANY LENGTH.                                *
                ;*    INPUT: HL=> PARAMETER BLOCK                       *
                ;*           PARAM+0,+1=ADDRESS OF OPERAND 1            *
                ;*           PARAM+2,+3=ADDRESS OF OPERAND 2            *
                ;*           PARAM+4=# OF BYTES 0-256                   *
                ;*    OUTPUT:OPERAND 1 LOCATION HOLDS RESULT            *
                ;***************************************************
                ;
7F00 F5         MPADDN   PUSH    AF              ;SAVE REGISTERS
7F01 C5                  PUSH    BC
7F02 D5                  PUSH    DE
7F03 E5                  PUSH    HL
7F04 DDE5                PUSH    IX
7F06 CD7F0A              CALL    0A7FH           ;***GET PB LOC'N***
7F09 E5                  PUSH    HL              ;TRANSFER TO IX
7F0A DDE1                POP     IX
7F0C DD5E00              LD      E,(IX+0)        ;GET OP 1 LOC'N
7F0F DD5601              LD      D,(IX+1)
7F12 DD6E02              LD      L,(IX+2)        ;GET OP 2 LOC'N
7F15 DD6603              LD      H,(IX+3)
7F18 DD4E04              LD      C,(IX+4)        ;GET # OF BYTES
7F1B 0600                LD      B,0             ;NOW IN BC
7F1D 0B                  DEC     BC              ;#-1
7F1E 09                  ADD     HL,BC           ;POINT TO LAST OP2
7F1F EB                  EX      DE,HL           ;SWAP DE AND HL
7F20 09                  ADD     HL,BC           ;POINT TO LAST OP1
7F21 EB                  EX      DE,HL           ;SWAP BACK
7F22 41                  LD      B,C             ;#-1 BACK TO B
7F23 04                  INC     B               ;ORIGINAL NUMBER
7F24 B7                  OR      A               ;CLEAR CARRY FOR FIRST ADD
7F25 1A         MPA010   LD      A,(DE)            ;GET OPERAND 1 BYTE
7F26 8E                  ADC     A,(HL)            ;ADD OPERAND 2
7F27 12                  LD      (DE),A            ;STORE RESULT
7F28 2B                  DEC     HL                ;POINT TO NEXT OP2
7F29 1B                  DEC     DE                ;POINT TO NEXT OP1
7F2A 10F9                DJNZ    MPA010            ;LOOP FOR N BYTES
7F2C DDE1                POP     IX              ;RESTORE REGISTERS
7F2E E1                  POP     HL
7F2F D1                  POP     DE
7F30 C1                  POP     BC
7F31 F1                  POP     AF
7F32 C9                  RET                     ;RETURN TO CALLING PROG
0000                     END
00000 TOTAL ERRORS
```

```
MPADDN DECIMAL VALUES

 245, 197, 213, 229, 221, 229, 205, 127, 10, 229,
 221, 225, 221, 94, 0, 221, 86, 1, 221, 110,
 2, 221, 102, 3, 221, 78, 4, 6, 0, 11,
 9, 235, 9, 235, 65, 4, 183, 26, 142, 18,
 43, 27, 16, 249, 221, 225, 225, 209, 193, 241,
 201

CHKSUM= 73
```

System Configuration

Model I, Model III, Model II Stand Alone.

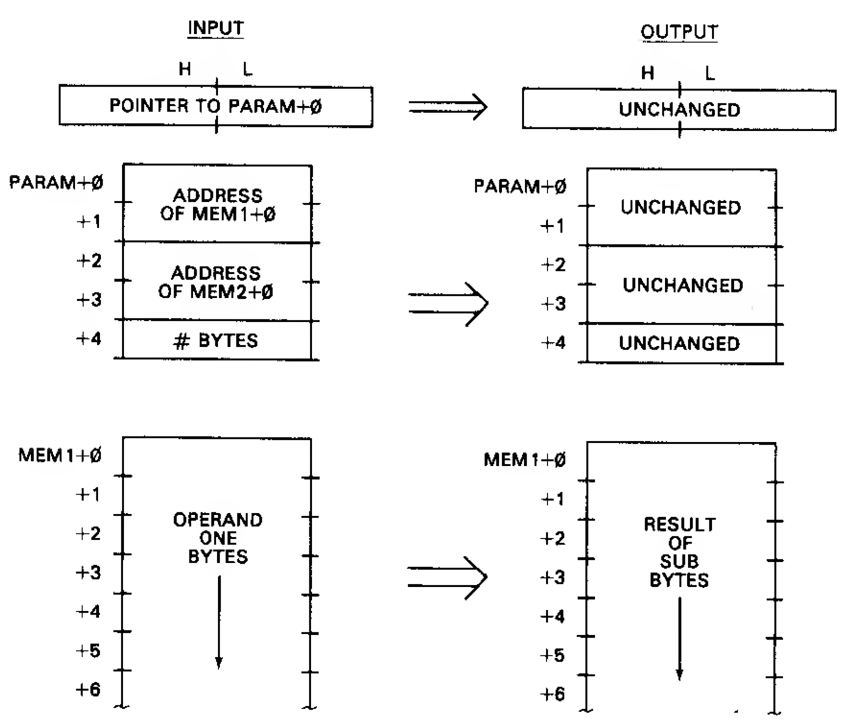
Description

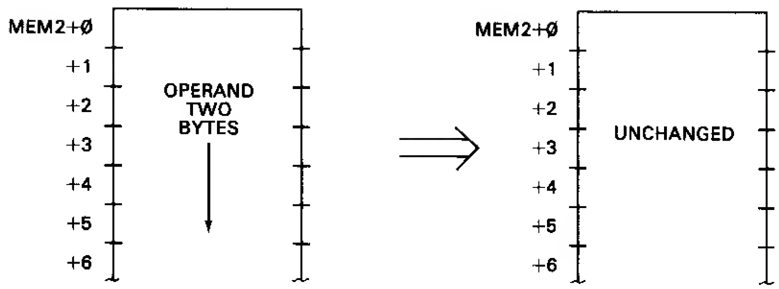
MPSUBT subtracts a "source" string of bytes from a "destination" string of bytes and puts the result of the subtract into the destination string. Each of the two strings is a multiple-precision binary number. Each of the two strings is assumed to be the same length. The length of each string may be any number from 1 through 255 or 0, which is 256 bytes.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the address of the destination string in standard Z-80 address format, least significant byte followed by most significant byte. The next two bytes of the parameter block contain the address of the source string in the same format. The next byte of the parameter block contains the number of bytes in the two operands.

On output, the parameter block and source string are unchanged. The destination string contains the result of the multiple-precision subtract.





Algorithm

The MPSUBT subroutine performs one subtract for each byte in the operands. The destination string address and source string address are first picked up from the parameter block and put into DE and HL, respectively. The number of bytes in the subtract is then picked up and put into the BC register pair. This number minus one is then added to the source and destination pointers so that they point to the least significant bytes of the source and destination strings. The number of bytes is then put into the B register for loop control.

The next destination byte is then picked up from the destination string (DE register pointer). An SBC is made of the two source string digits (HL register pointer). The result is then stored in the destination string.

The source and destination string pointers are then decremented by one to point to the next most significant two bytes of each operand. The B register count is then decremented by a DJNZ, and a loop back to MPS010 is made for the next subtract.

The carry is cleared before the first subtract, but successive subtracts subtract the carry from the preceding operation. If the destination operand was 00H, F5H, 6EH, 11H and the source operand was 00H, FFH, 77H, 33H, then the number of operand bytes must be 4. The result in the destination operand would be FFH, F5H, E6H, DEH. The result may be one bit larger than the original number of bits in the operands or may be a negative number.

Sample Calling Sequence

```
NAME OF SUBROUTINE? MPSUBT
HL VALUE? 4ØØØØ
PARAMETER BLOCK LOCATION? 4ØØØØ
PARAMETER BLOCK VALUES?
+ Ø  2  42ØØØ
+ 2  2  44ØØØ
+ 4  2  5  #OF BYTES
+ 6  Ø  Ø
MEMORY BLOCK 1 LOCATION? 42ØØØ
MEMORY BLOCK 1 VALUES?
+ Ø  1  Ø ┐
+ 1  1  Ø │
+ 2  1  Ø ├ DESTINATION = ØØØØØØØØH
+ 3  1  Ø │
+ 4  1  Ø ┘
+ 5  Ø  Ø
```

```
MEMORY BLOCK 2 LOCATION? 44000
MEMORY BLOCK 2 VALUES?
+ 0  1  0
+ 1  1  0
+ 2  1  0   SOURCE = 00000001H
+ 3  1  0
+ 4  1  1
+ 5  0  0
MOVE SUBROUTINE TO? 38000
SUBROUTINE EXECUTED AT  38000
INPUT:               OUTPUT:
HL= 40000            HL= 40000
PARAM+ 0   16        PARAM+ 0   16
PARAM+ 1   164       PARAM+ 1   164
PARAM+ 2   224       PARAM+ 2   224    UNCHANGED
PARAM+ 3   171       PARAM+ 3   171
PARAM+ 4   5         PARAM+ 4   5
PARAM+ 5   0         PARAM+ 5   0
MEMB1+ 0   0         MEMB1+ 0   255
MEMB1+ 1   0         MEMB1+ 1   255
MEMB1+ 2   0         MEMB1+ 2   255    RESULT = FFFFFFFFH
MEMB1+ 3   0         MEMB1+ 3   255
MEMB1+ 4   0         MEMB1+ 4   255
MEMB2+ 0   0         MEMB2+ 0   0
MEMB2+ 1   0         MEMB2+ 1   0
MEMB2+ 2   0         MEMB2+ 2   0      SOURCE UNCHANGED
MEMB2+ 3   0         MEMB2+ 3   0
MEMB2+ 4   1         MEMB2+ 4   1

NAME OF SUBROUTINE?
```

Notes

1. The destination string is a fixed length. Leading zero bytes must precede the operands to handle the result, which may be one bit larger than either of the operands.

2. This may be either a "signed" or "unsigned" subtract. If a two's complement number is used, then the sign must be "sign extended" to the more significant bits of the operands.

Program Listing

```
7F00                    ORG     7F00H           ;0522
           ;**********************************************************
           ;* MULTIPLE-PRECISION SUBTRACT. SUBTRACTS TWO MULTIPLE-  *
           ;* PRECISION OPERANDS, ANY LENGTH.                        *
           ;*     INPUT: HL=> PARAMETER BLOCK                        *
           ;*              PARAM+0,+1=ADDRESS OF OPERAND 1           *
           ;*              PARAM+2,+3=ADDRESS OF OPERAND 2           *
           ;*              PARAM+4=# OF BYTES 0-256                  *
           ;*     OUTPUT:OPERAND 1 LOCATION HOLDS RESULT             *
           ;**********************************************************
           ;
7F00 F5     MPSUBT      PUSH    AF              ;SAVE REGISTERS
7F01 C5                 PUSH    BC
7F02 D5                 PUSH    DE
7F03 E5                 PUSH    HL
7F04 DDE5               PUSH    IX
7F06 CD7F0A             CALL    0A7FH           ;***GET PB LOC'N***
7F09 E5                 PUSH    HL              ;TRANSFER TO IX
7F0A DDE1               POP     IX
7F0C DD5E00             LD      E,(IX+0)        ;GET OP 1 LOC'N
```

```
7F0F DD5601    00300           LD      D,(IX+1)
7F12 DD6E02    00310           LD      L,(IX+2)       ;GET OP 2 LOC'N
7F15 DD6603    00320           LD      H,(IX+3)
7F18 DD4E04    00330           LD      C,(IX+4)       ;GET # OF BYTES
7F1B 0600      00340           LD      B,0            ;NOW IN BC
7F1D 0B        00350           DEC     BC             ;#-1
7F1E 09        00360           ADD     HL,BC          ;POINT TO LAST OP2
7F1F EB        00370           EX      DE,HL          ;SWAP DE AND HL
7F20 09        00380           ADD     HL,BC          ;POINT TO LAST OP1
7F21 EB        00390           EX      DE,HL          ;SWAP BACK
7F22 41        00400           LD      B,C            ;#-1 BACK TO B
7F23 04        00410           INC     B              ;ORIGINAL NUMBER
7F24 B7        00420           OR      A              ;CLEAR CARRY FOR FIRST SUB
7F25 1A        00430 MPS010    LD      A,(DE)           ;GET OPERAND 1 BYTE
7F26 9E        00440           SBC     A,(HL)           ;SUB OPERAND 2
7F27 12        00450           LD      (DE),A           ;STORE RESULT
7F28 2B        00460           DEC     HL               ;POINT TO NEXT OP2
7F29 1B        00470           DEC     DE               ;POINT TO NEXT OP1
7F2A 10F9      00480           DJNZ    MPS010           ;LOOP FOR N BYTES
7F2C DDE1      00490           POP     IX             ;RESTORE REGISTERS
7F2E E1        00500           POP     HL
7F2F D1        00510           POP     DE
7F30 C1        00520           POP     BC
7F31 F1        00530           POP     AF
7F32 C9        00540           RET                    ;RETURN TO CALLING PROG
0000           00550           END
00000 TOTAL ERRORS
```

```
MPSUBT DECIMAL VALUES


245, 197, 213, 229, 221, 229, 205, 127, 10, 229,
221, 225, 221, 94, 0, 221, 86, 1, 221, 110,
2, 221, 102, 3, 221, 78, 4, 6, 0, 11,
9, 235, 9, 235, 65, 4, 183, 26, 158, 18,
43, 27, 16, 249, 221, 225, 225, 209, 193, 241,
201

CHKSUM= 89
```

MSLEFT: MULTIPLE SHIFT LEFT

System Configuration

Model I, Model III, Model II Stand Alone.

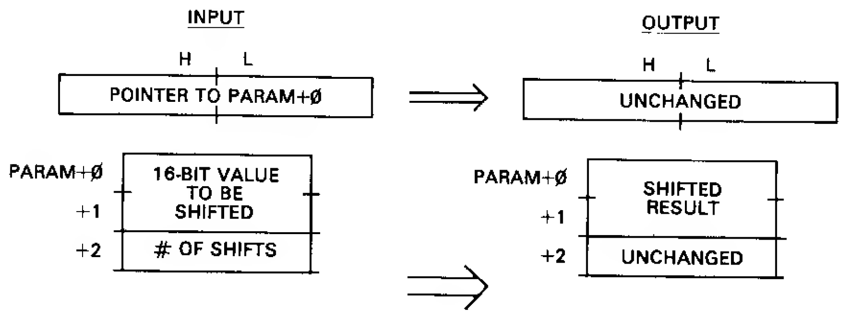
Description

MSLEFT shifts a given 16-bit value left a specified number of bit positions. The shift performed is a "logical" shift where zeroes fill vacated bit positions on the right.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the number to be shifted in standard Z-80 16-bit format, least significant byte followed by most significant byte. The next byte of the parameter block contains the number of shifts to be performed, from 1 to 15.

On output, the value in the first two bytes of the parameter block has been shifted the appropriate number of times. The count in the third byte of the parameter block remains unchanged.



Algorithm

The MSLEFT subroutine performs the shift by placing the number to be shifted in HL and the count in the B register. HL is added to itself a number of times corresponding to the count in the B register to effect the shift.

Sample Calling Sequence

```
NAME OF SUBROUTINE? MSLEFT
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0  2  1  VALUE TO BE SHIFTED = 0000000000000001
+ 2  1  8  8 SHIFTS
+ 3  0  0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 50000
SUBROUTINE EXECUTED AT  50000
INPUT:               OUTPUT:
HL= 40000            HL= 40000
PARAM+ 0   1         PARAM+ 0   0 ┐
PARAM+ 1   0         PARAM+ 1   1 ┘ RESULT = 0000000100000000
PARAM+ 2   8         PARAM+ 2   8   UNCHANGED

NAME OF SUBROUTINE?
```

Notes

1. If 0 is specified as a shift count, 256 shifts will be done, resulting in all zeroes in the result.
2. If 16 to 255 shifts are specified, the result will be all zeroes.
3. Note that the value to be shifted is ls bytes, ms byte.

Program Listing

```
7F00            ORG     7F00H           ;0522
        ;*********************************************************
        ;* MULTIPLE SHIFT LEFT. SHIFTS THE GIVEN 16-BIT VALUE    *
        ;* A SPECIFIED NUMBER OF SHIFTS IN LOGICAL FASHION       *
        ;*      INPUT: HL=>PARAMETER BLOCK                        *
        ;*             PARAM+0,+1=VALUE TO BE SHIFTED             *
        ;*             PARAM+2=NUMBER OF SHIFTS                   *
        ;*      OUTPUT:PARAM+0,+1=SHIFTED VALUE                   *
        ;*********************************************************
```

```
               00190 ;
7F00 C5        00200 MSLEFT  PUSH    BC              ;SAVE REGISTERS
7F01 E5        00210         PUSH    HL
7F02 DDE5      00220         PUSH    IX
7F04 CD7F0A    00230         CALL    0A7FH           ;***GET PB LOC'N***
7F07 E5        00240         PUSH    HL              ;TRANSFER TO IX
7F08 DDE1      00250         POP     IX
7F0A DD6E00    00260         LD      L,(IX+0)        ;GET LSB OF VALUE
7F0D DD6601    00270         LD      H,(IX+1)        ;GET MSB OF VALUE
7F10 DD4602    00280         LD      B,(IX+2)        ;GET # OF SHIFTS
7F13 29        00290 MSL010  ADD     HL,HL             ;LEFT SHIFT MS BYTE
7F14 10FD      00300         DJNZ    MSL010            ;LOOP 'TIL DONE
7F16 DD7500    00310 MSL030  LD      (IX+0),L        ;STORE SHIFTED RESULT
7F19 DD7401    00320         LD      (IX+1),H
7F1C DDE1      00330 MSL040  POP     IX              ;RESTORE REGISTERS
7F1E E1        00340         POP     HL
7F1F C1        00350         POP     BC
7F20 C9        00360         RET                     ;RETURN TO CALLING PROG
0000           00370         END
00000 TOTAL ERRORS
```

```
MSLEFT DECIMAL VALUES

197, 229, 221, 229, 205, 127, 10, 229, 221, 225,
221, 110, 0, 221, 102, 1, 221, 70, 2, 41,
16, 253, 221, 117, 0, 221, 116, 1, 221, 225,
225, 193, 201

CHKSUM= 28
```

MSRGHT: MULTIPLE SHIFT RIGHT

System Configuration

Model I, Model III, Model II Stand Alone.

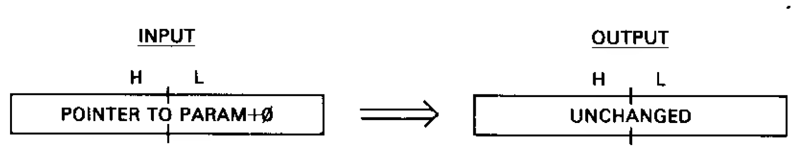
Description

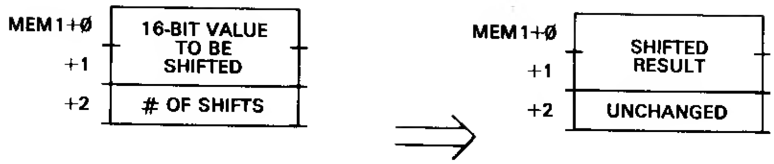
MSRGHT shifts a given 16-bit value right a specified number of bit positions. The shift performed is a "logical" shift where zeroes fill vacated bit positions on the left.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the number to be shifted in standard Z-80 16-bit format, least significant byte followed by most significant byte. The next byte of the parameter block contains the number of shifts to be performed, from 1 to 15.

On output, the value in the first two bytes of the parameter block has been shifted the appropriate number of times. The count in the third byte of the parameter block remains unchanged.





Algorithm

The MSRGHT subroutine performs the shift by placing the number to be shifted in HL and the count in the B register. HL is shifted right by first shifting H with an SRL. This shifts H one bit position, with the carry being set by the lsb of H. L is then shifted right by an RR, which shifts L to itself and places the previous value of the carry into the msb of L. This shift sequence is done a number of times corresponding to the count in the B register.

Sample Calling Sequence

```
NAME OF SUBROUTINE? MSRGHT
HL VALUE? 50000
PARAMETER BLOCK LOCATION? 50000
PARAMETER BLOCK VALUES?
+ 0  2  32768   VALUE TO BE SHIFTED = 1000000000000000
+ 2  1  15      15 SHIFTS
+ 3  0  0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 44444
SUBROUTINE EXECUTED AT  44444
INPUT:             OUTPUT:
HL= 50000          HL= 50000
PARAM+ 0  0        PARAM+ 0  1  ┐
PARAM+ 1  128      PARAM+ 1  0  ┘ RESULT = 0000000000000001
PARAM+ 2  15       PARAM+ 2  15   UNCHANGED

NAME OF SUBROUTINE?
```

Notes

1. If 0 is specified as a shift count, 256 shifts will be done, resulting in all zeroes in the result.

2. If 16 to 255 shifts are specified, the result will be all zeroes.

Program Listing

```
7F00                  ORG     7F00H           ;0522
            ;**********************************************************
            ;* MULTIPLE SHIFT RIGHT. SHIFTS THE GIVEN 16-BIT VALUE  *
            ;* A SPECIFIED NUMBER OF SHIFTS IN LOGICAL FASHION      *
            ;*     INPUT: HL=>PARAMETER BLOCK                       *
            ;*            PARAM+0,+1=VALUE TO BE SHIFTED            *
            ;*            PARAM+3=NUMBER OF SHIFTS                  *
            ;*     OUTPUT:PARAM+0,+1=SHIFTED VALUE                  *
            ;**********************************************************
            ;
7F00 C5     MSRGHT    PUSH    BC              ;SAVE REGISTERS
7F01 E5               PUSH    HL
7F02 DDE5             PUSH    IX
7F04 CD7F0A           CALL    0A7FH           ;***GET PB LOC'N***
```

```
7F07 E5        00240          PUSH   HL             ;TRANSFER TO IX
7F08 DDE1      00250          POP    IX
7F0A DD6E00    00260          LD     L,(IX+0)       ;GET LSB OF VALUE
7F0D DD6601    00270          LD     H,(IX+1)       ;GET MSB OF VALUE
7F10 DD4602    00280          LD     B,(IX+2)       ;GET # OF SHIFTS
7F13 CB3C      00290 MSR010   SRL    H                ;RIGHT SHIFT MS BYTE
7F15 CB1D      00300          RR     L                ;RIGHT SHIFT LS BYTE
7F17 10FA      00310          DJNZ   MSR010           ;LOOP 'TIL DONE
7F19 DD7500    00320 MSR030   LD     (IX+0),L       ;STORE SHIFTED RESULT
7F1C DD7401    00330          LD     (IX+1),H
7F1F DDE1      00340 MSR040   POP    IX             ;RESTORE REGISTERS
7F21 E1        00350          POP    HL
7F22 C1        00360          POP    BC
7F23 C9        00370          RET                   ;RETURN TO CALLING PROG
0000           00380          END
00000 TOTAL ERRORS
```

```
MSRGHT DECIMAL VALUES

197, 229, 221, 229, 205, 127, 10, 229, 221, 225,
221, 110, 0, 221, 102, 1, 221, 70, 2, 203,
60, 203, 29, 16, 250, 221, 117, 0, 221, 116,
1, 221, 225, 225, 193, 201

CHKSUM= 223
```

MUNOTE: MUSICAL NOTE ROUTINE

System Configuration

Model I, Model III.

Description

MUNOTE outputs a musical note through the cassette port. The cassette jack output may be connected to a small, inexpensive amplifier for music, audio sound effects, or warning tones. The tone ranges over seven octaves starting with A three octaves below middle A and ending with G#, three octaves above middle G#. The duration of the tone may be specified by the user in 1/16th second increments. Pitches and durations are approximate!

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the address of MUNOTE in standard Z-80 address format, least significant byte followed by most significant byte. This address may be easily picked up from the USR call if MUNOTE is called from BASIC or from the assembly-language CALL address. It is necessary so that the code in MUNOTE is completely relocatable. The next byte of the parameter block contains the note value of 0 through 83. This note value corresponds to musical notes as shown in the table below. The next byte of the parameter block specifies the duration of the note in 1/16th second increments. A value of 3, for example, would be 3/16ths second.

On output, the contents of the parameter block remain unchanged and the note has been played.

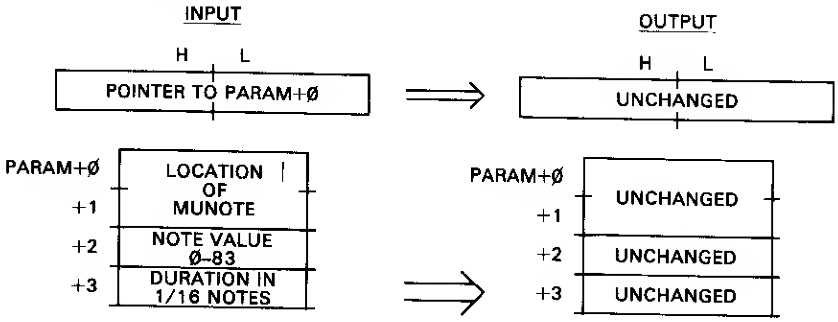


Table of values for musical notes.

VAL	NOTE	FREQUENCY	TABLE VALUES	
0	A	27.5	122, 5	1, 0
1	A#	29.1352	43, 5	1, 0
2	B	30.8677	225, 4	1, 0
3	C	32.7032	154, 4	2, 0
4	C#	34.6478	88, 4	2, 0
5	D	36.7081	26, 4	2, 0
6	D#	38.8909	223, 3	2, 0
7	E	41.2035	167, 3	2, 0
8	F	43.6535	114, 3	2, 0
9	F#	46.2493	65, 3	2, 0
10	G	48.9995	18, 3	3, 0
11	G#	51.9131	230, 2	3, 0
12	A	55	188, 2	3, 0
13	A#	58.2705	148, 2	3, 0
14	B	61.7355	111, 2	3, 0
15	C	65.4064	76, 2	4, 0
16	C#	69.2957	43, 2	4, 0
17	D	73.4163	12, 2	4, 0
18	D#	77.7818	238, 1	4, 0
19	E	82.407	210, 1	5, 0
20	F	87.3071	184, 1	5, 0
21	F#	92.4987	159, 1	5, 0
22	G	97.999	136, 1	6, 0
23	G#	103.826	114, 1	6, 0
24	A	110	93, 1	6, 0
25	A#	116.541	73, 1	7, 0
26	B	123.471	54, 1	7, 0
27	C	130.813	37, 1	8, 0
28	C#	138.592	20, 1	8, 0
29	D	146.833	5, 1	9, 0
30	D#	155.564	246, 0	9, 0
31	E	164.814	232, 0	10, 0
32	F	174.614	219, 0	10, 0
33	F#	184.997	206, 0	11, 0
34	G	195.998	195, 0	12, 0
35	G#	207.653	184, 0	12, 0
36	A	220	173, 0	13, 0
37	A#	233.082	163, 0	14, 0
38	B	246.942	154, 0	15, 0
39	C	261.626	145, 0	16, 0
40	C#	277.183	137, 0	17, 0
41	D	293.665	129, 0	18, 0
42	D#	311.128	122, 0	19, 0
43	E	329.628	115, 0	20, 0
44	F	349.229	108, 0	21, 0
45	F#	369.995	102, 0	23, 0
46	G	391.996	96, 0	24, 0
47	G#	415.306	91, 0	25, 0

48	A	440.001	86, 0	27, 0
49	A#	466.165	81, 0	29, 0
50	B	493.884	76, 0	30, 0
51	C	523.252	72, 0	32, 0
52	C#	554.367	67, 0	34, 0
53	D	587.331	64, 0	36, 0
54	D#	622.256	60, 0	38, 0
55	E	659.257	56, 0	41, 0
56	F	698.458	53, 0	43, 0
57	F#	739.991	50, 0	46, 0
58	G	783.993	47, 0	48, 0
59	G#	830.612	44, 0	51, 0
60	A	880.003	42, 0	55, 0
61	A#	932.33	39, 0	58, 0
62	B	987.769	37, 0	61, 0
63	C	1046.51	35, 0	65, 0
64	C#	1108.73	33, 0	69, 0
65	D	1174.66	31, 0	73, 0
66	D#	1244.51	29, 0	77, 0
67	E	1318.51	27, 0	82, 0
68	F	1396.92	25, 0	87, 0
69	F#	1479.98	24, 0	92, 0
70	G	1567.99	22, 0	97, 0
71	G#	1661.22	21, 0	103, 0
72	A	1760.01	20, 0	110, 0
73	A#	1864.66	18, 0	116, 0
74	B	1975.54	17, 0	123, 0
75	C	2093.01	16, 0	130, 0
76	C#	2217.47	15, 0	138, 0
77	D	2349.33	14, 0	146, 0
78	D#	2489.03	13, 0	155, 0
79	E	2637.03	12, 0	164, 0
80	F	2793.84	12, 0	174, 0
81	F#	2959.97	11, 0	184, 0
82	G	3135.98	10, 0	195, 0
83	G#	3322.45	9, 0	207, 0

Algorithm

Operation of MUNOTE is very similar to TONOUT. MUNOTE, however, picks up a frequency count and duration count from the MUNTB table. This table is referenced to the note value in the parameter block. The note value of 0 through 83 is multiplied by 4, added to the starting address of MUNOTE from the parameter block, and then added to the displacement of the table, MUNTB, to point to the table entry. The frequency count and duration count from MUNTB are then picked up and put into DE and BC, respectively. The duration count is multiplied by the number of 16ths specified in the parameter block, and the final duration count is put into IX. From this point on, the code is almost identical to the TONOUT code.

MUNOTE uses two loops. The outer loop (from MUN010) produces the number of cycles equal to the duration count. The inner loop is made up of two parts. The MUN020 portion outputs an "on" pulse from the cassette output. The MUN030 portion turns off the cassette port for the same period of time. Both portions use the frequency count from the DE register for a timing loop count.

The MUN010 loop puts the DE frequency count into HL and turns on the cassette (OUT 0FFH,A). The count in HL is then decremented by one in the MUN020 timing loop. At the end of the loop, the count is again put into HL from DE, the cassette is turned off, and the count is decremented by one in the MUN030 timing loop. After this loop, the duration, or cycle, count in IX is decremented by one and if it is not negative, a jump is made back to MUN010 for the next cycle.

Sample Calling Sequence

```
NAME OF SUBROUTINE? MUNOTE
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0   2   37000  START OF MUNOTE
+ 2   1   60     FIFTH OCTAVE, A
+ 3   1   2      1/8TH SECOND
+ 4   0   0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 37000
SUBROUTINE EXECUTED AT  37000
INPUT:               OUTPUT:
HL= 40000            HL= 40000
PARAM+ 0   136       PARAM+ 0   136
PARAM+ 1   144       PARAM+ 1   144
PARAM+ 2   60        PARAM+ 2   60
PARAM+ 3   2         PARAM+ 3   2

NAME OF SUBROUTINE?
```

Notes

1. The table values are for a standard TRS-80 Model I clock frequency. They must be recomputed for clock speed upgrades or adjusted for a Model III. Multiply the frequency values by 1.143 and divide the duration values by 1.143 for a Model III.

2. Lower octave durations and higher octave frequencies are approximate.

Program Listing

```
7F00                      ORG     7F00H           ;0522
                 ;*******************************************************
                 ;* MUSICAL NOTE ROUTINE. OUTPUTS MUSICAL NOTE THROUGH  *
                 ;* CASSETTE PORT.                                      *
                 ;*     INPUT: HL=> PARAMETER BLOCK                     *
                 ;*                PARAM+0,+1=LOCATION OF MUNOTE        *
                 ;*                PARAM+2=NOTE VALUE, 0 THROUGH 83     *
                 ;*                PARAM+3=DURATION IN 1/16TH NOTES     *
                 ;*     OUTPUT:NOTE OUTPUT TO CASSETTE PORT             *
                 ;*******************************************************
                 ;
7F00 F5          MUNOTE   PUSH    AF              ;SAVE REGISTERS
7F01 C5                   PUSH    BC
7F02 D5                   PUSH    DE
7F03 E5                   PUSH    HL
7F04 DDE5                 PUSH    IX
7F06 FDE5                 PUSH    IY
7F08 CD7F0A               CALL    0A7FH           ;***GET PB LOC'N***
7F0B E5                   PUSH    HL              ;TRANSFER TO IX
7F0C DDE1                 POP     IX
7F0E DD6E02               LD      L,(IX+2)        ;GET NOTE VALUE
7F11 2600                 LD      H,0             ;NOW IN HL
```

```
7F13 29         00320           ADD     HL,HL           ;INDEX*2
7F14 29         00330           ADD     HL,HL           ;INDEX*4
7F15 DD5E00     00340           LD      E,(IX+0)        ;PUT MUNOTE BASE IN BC
7F18 DD5601     00350           LD      D,(IX+1)
7F1B 19         00360           ADD     HL,DE           ;BASE PLUS INDEX
7F1C 115F00     00370           LD      DE,MUNTB        ;TABLE DISPLACEMENT
7F1F 19         00380           ADD     HL,DE           ;POINT TO ENTRY
7F20 E5         00390           PUSH    HL              ;TRANSFER ENTRY LOC TO IY
7F21 FDE1       00400           POP     IY
7F23 FD5E00     00410           LD      E,(IY+0)        ;PUT FREQ COUNT IN DE
7F26 FD5601     00420           LD      D,(IY+1)
7F29 FD4E02     00430           LD      C,(IY+2)        ;PUT DUR COUNT IN BC
7F2C FD4603     00440           LD      B,(IY+3)
7F2F 210000     00450           LD      HL,0            ;INITIALIZE DURATION
7F32 DD7E03     00460           LD      A,(IX+3)        ;GET DURATION IN 1/16THS
7F35 09         00470 MUN005    ADD     HL,BC             ;CHANGE TO SPEC DURATION
7F36 3D         00480           DEC     A                 ;DECREMENT 1/16THS CNT
7F37 20FC       00490           JR      NZ,MUN005         ;LOOP TIL DONE
7F39 E5         00500 MUN008    PUSH    HL              ;TRANSFER NEW CNT TO IX
7F3A DDE1       00510           POP     IX
7F3C 01FFFF     00520           LD      BC,-1           ;FOR TIGHT LOOP
7F3F 6B         00530 MUN010    LD      L,E               ;PUT FREQ COUNT IN HL 4
7F40 62         00540           LD      H,D               ;4
7F41 3E01       00550           LD      A,1               ;MAXIMUM POSITIVE 7
7F43 D3FF       00560           OUT     (0FFH),A          ;OUTPUT 11
7F45 09         00570 MUN020    ADD     HL,BC               ;COUNT-1 11
7F46 DA457F     00580           JP      C,MUN020            ;LOOP FOR 1/2 CYCLE 7/12
7F49 6B         00590           LD      L,E               ;PUT FREQ COUNT IN HL 4
7F4A 62         00600           LD      H,D               ;4
7F4B 3E02       00610           LD      A,2               ;MAXIMUM NEGATIVE 7
7F4D D3FF       00620           OUT     (0FFH),A          ;OUTPUT 11
7F4F 09         00630 MUN030    ADD     HL,BC               ;COUNT-1 11
7F50 38FD       00640           JR      C,MUN030            ;LOOP FOR 1/2 CYCLE 7/12
7F52 DD09       00650           ADD     IX,BC             ;DECREMENT DUR COUNT 15
7F54 38E9       00660           JR      C,MUN010          ;LOOP IF NOT DONE 7/12
7F56 FDE1       00670           POP     IY              ;RESTORE REGISTERS
7F58 DDE1       00680           POP     IX
7F5A E1         00690           POP     HL
7F5B D1         00700           POP     DE
7F5C C1         00710           POP     BC
7F5D F1         00720           POP     AF
7F5E C9         00730           RET                     ;RETURN TO CALLING PROG
005F            00740 MUNTB     EQU     $-MUNOTE
                00750 ; MUSICAL NOTE TABLE. ENTRY+0,+1 IS FREQUENCY COUNT.
                00760 ; ENTRY+2,+3 IS DURATION COUNT FOR 1/16THS.
0000            00770           END
00000 TOTAL ERRORS
```

MUNOTE DECIMAL VALUES

```
245, 197, 213, 229, 221, 229, 253, 229, 205, 127,
10, 229, 221, 225, 221, 110, 2, 38, 0, 41,
41, 221, 94, 0, 221, 86, 1, 25, 17, 95,
0, 25, 229, 253, 225, 253, 94, 0, 253, 86,
1, 253, 78, 2, 253, 70, 3, 33, 0, 0,
221, 126, 3, 9, 61, 32, 252, 229, 221, 225,
1, 255, 255, 107, 98, 62, 1, 211, 255, 9,
218, 69, 127, 107, 98, 62, 2, 211, 255, 9,
56, 253, 221, 9, 56, 233, 253, 225, 221, 225,
225, 209, 193, 241, 201
```

CHKSUM= 225

System Configuration

Model I, Model III.

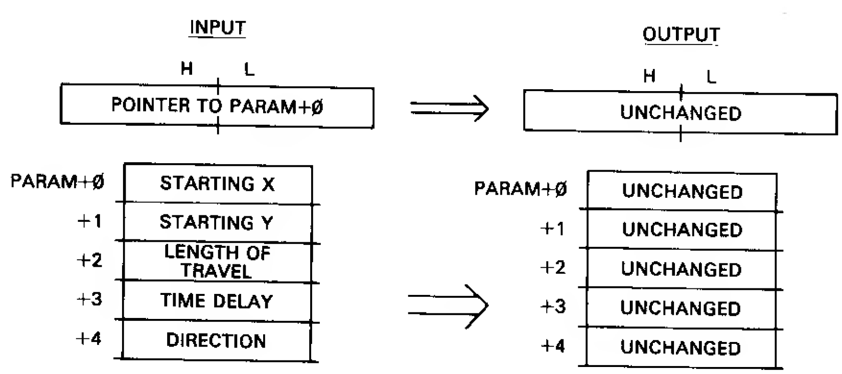
Description

MVDIAG moves a "dot" along a diagonal line with a varying time delay. This effect can be used for games or other applications. The dot may move along the diagonal from "bottom" to "top" of the screen, or from "top" to "bottom." The amount of time that the dot remains in any position can be adjusted under program control.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first byte of the parameter block contains the starting x character position of the dot, from 0 to 63. The next byte of the parameter block contains the starting line number y of the dot, from 0 to 15. The next byte of the parameter block contains the number of character positions of travel. This will be a maximum of 16 for a diagonal that starts 16 character positions or greater from the side of the screen. The next byte of the parameter block contains the time delay value from 1 to 255 or 0 (256). One is a minimum time delay, while 255 and 0 (256) are maximum time delays. The next byte of the parameter block contains the direction of travel—0 is up to the right, 1 is up to the left, 2 is down to the right, and 3 is down to the left.

On output, the parameter block contents are unchanged. The dot has moved over the specified diagonal.



Algorithm

The MVDIAG subroutine performs the move by computing the starting address of the dot in video display memory, by computing the "increment" to add to the address to obtain the next dot position, and by controlling the move with a count of the number of character positions involved.

First, the line number value is picked up from the parameter block. This is multiplied by 64 to find the number of bytes (displacement) from the start of video display memory. This value is added to 3C00H to find the actual video memory address for the line start. This value is added to the character position of the start from the parameter block to find the starting position in video display memory.

Next, a test is made of the direction of travel. Based on the direction, an increment value of −41H (up to left), −3FH (up to right), 3FH (down to left), or 41H (down to right) is found. This represents the number to be added to the last video display memory location to find the next video display memory location for the dot.

The code at MVD020 is the main loop of the subroutine. A byte of 0BFH is stored to the current video display memory position. A time delay is then done by decrementing the count value in the C register. After the delay, a byte of 80H is stored to "erase" the last dot.

The increment value is then added to the current video display memory position to find the next location of the dot. A count of the number of character positions involved is then decremented, and a jump is made to MVD020 if the count is not zero.

Sample Calling Sequence

```
NAME OF SUBROUTINE? MVDIAG
HL VALUE? 43333
PARAMETER BLOCK LOCATION? 43333
PARAMETER BLOCK VALUES?
+ 0  1  8     X=8
+ 1  1  15    Y=15
+ 2  1  16    LENGTH = 16 (END X, Y = 24, 0)
+ 3  1  0     MAXIMUM DELAY
+ 4  1  0     UP TO RIGHT
+ 5  0  0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 38888
SUBROUTINE EXECUTED AT  38888
INPUT:              OUTPUT:
HL= 43333           HL= 43333
PARAM+ 0   8        PARAM+ 0   8  ┐
PARAM+ 1   15       PARAM+ 1   15 │
PARAM+ 2   16       PARAM+ 2   16 ├UNCHANGED
PARAM+ 3   0        PARAM+ 3   0  │
PARAM+ 4   0        PARAM+ 4   0  ┘

NAME OF SUBROUTINE?
```

Notes

1. The program may "bomb" the system if the length of travel goes beyond video display memory boundaries or if x or y are incorrect values. Maximum length is 16.

2. Add additional time wasting instructions as required.

3. Delete time wasting instructions as required. Substituting NOPs (zeroes) will shorten the delay.

4. Speed at maximum delay is about 85 character positions per second.

```
7F00                00100            ORG     7F00H           ;0522
                    00110 ;*****************************************************
                    00120 ;* MOVING DOT DIAGONAL. MOVES DOT ALONG DIAGONAL LINE   *
                    00130 ;* WITH VARYING TIME DELAY                             *
                    00140 ;*     INPUT: HL=> PARAMETER BLOCK                      *
                    00150 ;*            PARAM+0=STARTING CHAR POS'N (X)           *
                    00160 ;*            PARAM+1=STARTING LINE # (Y)               *
                    00170 ;*            PARAM+2=LENGTH OF TRAVEL IN CHAR POSNS    *
                    00180 ;*            PARAM+3=TIME DELAY, 1=MIN 255/0=MAX       *
                    00190 ;*            PARAM+4=0 IS UP TO RIGHT, 1 IS UP TO LEFT *
                    00200 ;*                    2 IS DOWN TO RIGHT, 3 IS DOWN TO  *
                    00210 ;*                    LEFT                              *
                    00220 ;*    OUTPUT:DOT MOVES ALONG DIAGONAL LINE              *
                    00230 ;*****************************************************
                    00240 ;
7F00 F5             00250 MVDIAG  PUSH    AF              ;SAVE REGISTERS
7F01 C5             00260         PUSH    BC
7F02 D5             00270         PUSH    DE
7F03 E5             00280         PUSH    HL
7F04 DDE5           00290         PUSH    IX
7F06 FDE5           00300         PUSH    IY
7F08 CD7F0A         00310         CALL    0A7FH           ;***GET PB LOC'N***
7F0B E5             00320         PUSH    HL              ;TRANSFER TO IX
7F0C DDE1           00330         POP     IX
7F0E 0606           00340         LD      B,6             ;ITERATION COUNT
7F10 DD6E01         00350         LD      L,(IX+1)        ;GET LINE #
7F13 2600           00360         LD      H,0             ;NOW IN HL
7F15 29             00370 MVD010  ADD     HL,HL             ;LINE# * 64
7F16 10FD           00380         DJNZ    MVD010            ;LOOP 'TIL DONE
7F18 01003C         00390         LD      BC,3C00H        ;START OF SCREEN
7F1B 09             00400         ADD     HL,BC           ;FIND LOC OF LINE START
7F1C DD4E00         00410         LD      C,(IX+0)        ;GET CHAR POSN (X)
7F1F 0600           00420         LD      B,0             ;NOW IN BC
7F21 09             00430         ADD     HL,BC           ;FIND ACTUAL LOC'N
7F22 DD4602         00440         LD      B,(IX+2)        ;GET LENGTH OF TRAVEL
7F25 DD4E04         00450         LD      C,(IX+4)        ;GET DIRECTION CODE
7F28 CB49           00460         BIT     1,C             ;TEST DIRECTION
7F2A 11BFFF         00470         LD      DE,-41H         ;INCREMENT FOR NEXT DOT
7F2D 2803           00480         JR      Z,MVD015        ;GO IF UP
7F2F 113F00         00490         LD      DE,3FH          ;INCREMENT FOR DOWN
7F32 CB41           00500 MVD015  BIT     0,C             ;TEST RIGHT/LEFT
7F34 2002           00510         JR      NZ,MVD020       ;GO IF LEFT
7F36 13             00520         INC     DE              ;RIGHT
7F37 13             00530         INC     DE
7F38 36BF           00540 MVD020  LD      (HL),0BFH         ;SET CHAR POS TO ALL ON
7F3A DD4E03         00550         LD      C,(IX+3)          ;GET DELAY COUNT
7F3D 0D             00560 MVD030  DEC     C                   ;DECREMENT COUNT
7F3E FD2A0000       00570         LD      IY,(0)              ;WASTE TIME
7F42 FD2A0000       00580         LD      IY,(0)
7F46 FD2A0000       00590         LD      IY,(0)
7F4A FD2A0000       00600         LD      IY,(0)
7F4E 20ED           00610         JR      NZ,MVD030           ;DELAY LOOP
7F50 3680           00620         LD      (HL),80H          ;RESET CHAR POS
7F52 19             00630         ADD     HL,DE             ;POINT TO NEXT POSITION
7F53 10E3           00640         DJNZ    MVD020            ;LOOP FOR LENGTH OF LINE
7F55 FDE1           00650         POP     IY
7F57 DDE1           00660         POP     IX              ;RESTORE REGISTERS
7F59 E1             00670         POP     HL
7F5A D1             00680         POP     DE
7F5B C1             00690         POP     BC
7F5C F1             00700         POP     AF
7F5D C9             00710         RET                     ;RETURN TO CALLING PROG
0000                00720         END
00000 TOTAL ERRORS
```

MVDIAG DECIMAL VALUES

```
245, 197, 213, 229, 221, 229, 253, 229, 205, 127,
10, 229, 221, 225, 6, 6, 221, 110, 1, 38,
0, 41, 16, 253, 1, 0, 60, 9, 221, 78,
0, 6, 0, 9, 221, 70, 2, 221, 78, 4,
203, 73, 17, 191, 255, 40, 3, 17, 63, 0,
203, 65, 32, 2, 19, 19, 54, 191, 221, 78,
3, 13, 253, 42, 0, 0, 253, 42, 0, 0,
253, 42, 0, 0, 253, 42, 0, 0, 32, 237,
54, 128, 25, 16, 227, 253, 225, 221, 225, 225,
209, 193, 241, 201
```

CHKSUM= 175

MVHORZ: MOVING DOT HORIZONTAL

System Configuration

Model I, Model III.

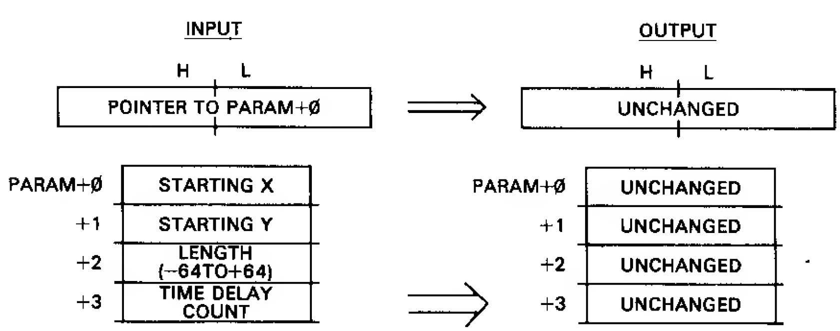
Description

MVHORZ moves a "dot" along a horizontal line with a varying time delay. This effect can be used for games or other applications. The dot may move along the horizontal line from right to left, or from left to right, on the screen. The amount of time that the dot remains in any position can be adjusted under program control.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first byte of the parameter block contains the starting x character position of the dot, from 0 to 63. The next byte of the parameter block contains the starting line number y of the dot, from 0 to 15. The next byte of the parameter block contains the number of character positions of travel. This will be a maximum of 64 for horizontal travel that starts at a right or left edge of the screen. The next byte of the parameter block contains the time delay value from 1 to 255 or 0 (256). One is a minimum time delay, while 255 and 0 (256) are maximum time delays.

On output, the parameter block contents are unchanged. The dot has moved over the specified horizontal line.



Algorithm

The MVHORZ subroutine performs the move by computing the starting address of the dot in video display memory, by finding the direction of travel, and by controlling the move with a count of the number of character positions involved.

First, the line number value is picked up from the parameter block. This is multiplied by 64 to find the number of bytes (displacement) from the start of video display memory. This value is added to 3C00H to find the actual video memory address for the line start. This value is added to the character position of the start from the parameter block to find the starting position in video display memory.

Next, a test is made of the direction of travel. Based on the direction, a "move right" code segment (MVH040) or a "move left" code segment (MVH020) is entered. Both segments are very similar, except that the "move right" increments the next character position pointer, while the "move left" decrements the next character position pointer.

In each code segment, a byte of 0BFH is stored to the current video display memory position. A time delay is then done by decrementing the count value in the C register. After the delay, a byte of 80H is stored to "erase" the last dot.

The current video display memory position in HL is then incremented or decremented to find the next location of the dot. The count of the number of character positions involved is then decremented, and a jump is made to MVH020 or MVH040 if the count is not zero.

Sample Calling Seqence

```
NAME OF SUBROUTINE? MVHORZ
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0   1   0   X=0
+ 1   1   8   Y=8
+ 2   1   64  LENGTH = 64 (END X, Y = 64, 8), RIGHT
+ 3   1   0   MAXIMUM DELAY
+ 4   0   0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 37000
SUBROUTINE EXECUTED AT  37000
INPUT:                OUTPUT:
HL= 40000             HL= 40000
PARAM+ 0   0          PARAM+ 0   0
PARAM+ 1   8          PARAM+ 1   8
PARAM+ 2   64         PARAM+ 2   64
PARAM+ 3   0          PARAM+ 3   0

NAME OF SUBROUTINE?
```

Notes

1. The program may "bomb" the system if the length of travel goes beyond video display memory boundaries. Maximum length is −64 or +64.

2. The program may "bomb" the system if the x and y coordinates are improperly specified.

3. Use additional time-wasting instructions as required.

4. Delete time-wasting instructions as required. NOPs (all zeroes) may be substituted to shorten delay times.

5. Speed at maximum delay is about 85 character positions per second.

Program Listing

```
7F00                          ORG     7F00H           ;0522
                  ;*******************************************************
                  ;* MOVING DOT HORIZONTAL. MOVES DOT ALONG HORIZONTAL   *
                  ;* LINE WITH VARYING TIME DELAY.                       *
                  ;*     INPUT: HL=> PARAMETER BLOCK                     *
                  ;*            PARAM+0=STARTING CHAR POS'N (X)          *
                  ;*            PARAM+1=STARTING LINE # (Y)              *
                  ;*            PARAM+2=LENGTH OF TRAVEL IN CHAR POSNS   *
                  ;*                  + IS TO RIGHT, - IS TO LEFT        *
                  ;*            PARAM+3=TIME DELAY, 1=MIN 255/0=MAX      *
                  ;*     OUTPUT:DOT MOVES ALONG LINE                     *
                  ;*******************************************************
                  ;
7F00 F5           MVHORZ  PUSH    AF              ;SAVE REGISTERS
7F01 C5                   PUSH    BC
7F02 E5                   PUSH    HL
7F03 DDE5                 PUSH    IX
7F05 FDE5                 PUSH    IY
7F07 CD7F0A               CALL    0A7FH           ;***GET PB LOC'N***
7F0A E5                   PUSH    HL              ;TRANSFER TO IX
7F0B DDE1                 POP     IX
7F0D 0606                 LD      B,6             ;ITERATION COUNT
7F0F DD6E01               LD      L,(IX+1)        ;GET LINE #
7F12 2600                 LD      H,0             ;NOW IN HL
7F14 29           MVH010  ADD     HL,HL             ;LINE# * 64
7F15 10FD                 DJNZ    MVH010            ;LOOP 'TIL DONE
7F17 01003C               LD      BC,3C00H        ;START OF SCREEN
7F1A 09                   ADD     HL,BC           ;FIND LOC OF LINE START
7F1B DD4E00               LD      C,(IX+0)        ;GET CHAR POSN (X)
7F1E 0600                 LD      B,0             ;NOW IN BC
7F20 09                   ADD     HL,BC           ;FIND ACTUAL LOC'N
7F21 DD4602               LD      B,(IX+2)        ;GET LENGTH OF TRAVEL
7F24 CB78                 BIT     7,B             ;TEST SIGN
7F26 2823                 JR      Z,MVH040        ;GO IF RIGHT
7F28 78                   LD      A,B             ;LEFT
7F29 ED44                 NEG                     ;FIND ABSOLUTE VALUE
7F2B 47                   LD      B,A             ;BACK TO B FOR DJNZ
7F2C 36BF         MVH020  LD      (HL),0BFH         ;SET CHAR POS TO ALL ON
7F2E DD4E03               LD      C,(IX+3)          ;GET DELAY COUNT
7F31 0D           MVH030  DEC     C                   ;DECREMENT COUNT
7F32 FD2A0000             LD      IY,(0)              ;WASTE TIME
7F36 FD2A0000             LD      IY,(0)
7F3A FD2A0000             LD      IY,(0)
7F3E FD2A0000             LD      IY,(0)
7F42 20ED                 JR      NZ,MVH030           ;DELAY LOOP
7F44 3680                 LD      (HL),80H          ;RESET CHAR POS
7F46 2B                   DEC     HL                ;POINT TO NEXT POSN
7F47 10E3                 DJNZ    MVH020            ;LOOP FOR LENGTH OF LINE
7F49 181D                 JR      MVH090          ;GO TO CLEAN UP
7F4B 36BF         MVH040  LD      (HL),0BFH         ;SET CHAR POS TO ALL ON
7F4D DD4E03               LD      C,(IX+3)          ;GET DELAY COUNT .
7F50 0D           MVH050  DEC     C                   ;DECREMENT COUNT
7F51 FD2A0000             LD      IY,(0)              ;WASTE TIME
7F55 FD2A0000             LD      IY,(0)
```

```
7F59 FD2A0000 00640          LD      IY,(0)
7F5D FD2A0000 00650          LD      IY,(0)
7F61 20ED     00660          JR      NZ,MVH050        ;DELAY LOOP
7F63 3680     00670          LD      (HL),80H         ;RESET CHAR POS
7F65 23       00680          INC     HL               ;POINT TO NEXT POSN
7F66 10E3     00690          DJNZ    MVH040           ;LOOP FOR LENGTH OF LINE
7F68 FDE1     00700 MVH090   POP     IY               ;RESTORE REGISTERS
7F6A DDE1     00710          POP     IX
7F6C E1       00720          POP     HL
7F6D C1       00730          POP     BC
7F6E F1       00740          POP     AF
7F6F C9       00750          RET                      ;RETURN TO CALLING PROG
0000          00760          END
00000 TOTAL ERRORS
```

```
MVHORZ DECIMAL VALUES

245, 197, 229, 221, 229, 253, 229, 205, 127, 10,
229, 221, 225, 6, 6, 221, 110, 1, 38, 0,
41, 16, 253, 1, 0, 60, 9, 221, 78, 0,
6, 0, 9, 221, 70, 2, 203, 120, 40, 35,
120, 237, 68, 71, 54, 191, 221, 78, 3, 13,
253, 42, 0, 0, 253, 42, 0, 0, 253, 42,
0, 0, 253, 42, 0, 0, 32, 237, 54, 128,
43, 16, 227, 24, 29, 54, 191, 221, 78, 3,
13, 253, 42, 0, 0, 253, 42, 0, 0, 253,
42, 0, 0, 253, 42, 0, 0, 32, 237, 54,
128, 35, 16, 227, 253, 225, 221, 225, 225, 193,
241, 201

CHKSUM= 146
```

MVVERT: MOVING DOT VERTICAL

System Configuration

Model I, Model III.

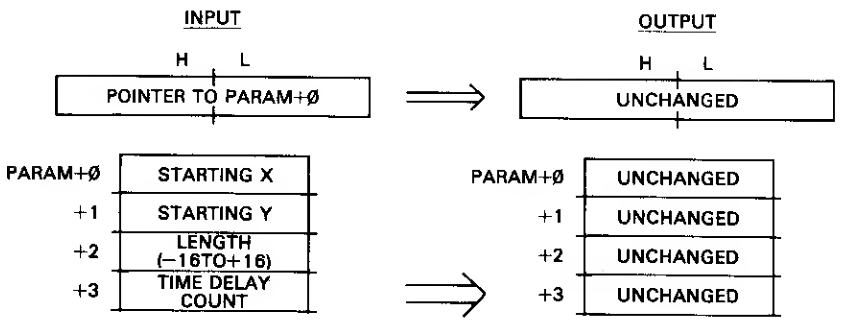
Description

MVVERT moves a "dot" along a vertical line with a varying time delay. This effect can be used for games or other applications. The dot may move along the vertical line from top to bottom, or from bottom to top, on the screen. The amount of time that the dot remains in any position can be adjusted under program control.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first byte of the parameter block contains the starting x character position of the dot, from 0 to 63. The next byte of the parameter block contains the starting line number y of the dot, from 0 to 15. The next byte of the parameter block contains the number of character positions of travel. This will be a maximum of 16 for vertical travel that starts at the top or bottom of the screen. The next byte of the parameter block contains the time delay value from 1 to 255 or 0 (256). One is a minimum time delay, while 255 and 0 (256) are maximum time delays.

On output, the parameter block contents are unchanged. The dot has moved over the specified vertical line.



Algorithm

The MVVERT subroutine performs the move by computing the starting address of the dot in video display memory, by finding the direction of travel, and by controlling the move with a count of the number of character positions involved.

First, the line number value is picked up from the parameter block. This is multiplied by 64 to find the number of bytes (displacement) from the start of video display memory. This value is added to 3C00H to find the actual video memory address for the line start. This value is added to the character position of the start from the parameter block to find the starting position in video display memory.

Next, a test is made of the direction of travel. Based on the direction, an increment value of 40H (down) or −40H (up) is stored in DE.

The code at MVV020 is the main loop of the subroutine. A byte of 0BFH is stored to the current video display memory position. A time delay is then done by decrementing the count value in the C register. After the delay, a byte of 80H is stored to "erase" the last dot.

The current video display memory position in HL is then incremented or decremented by the increment value in DE to find the next location of the dot. The count of the number of character positions involved is then decremented, and a jump is made to MVV020.

Sample Calling Sequence

```
NAME OF SUBROUTINE? MVVERT
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0   1   32     X = 32
+ 1   1   0      Y = 0
+ 2   1   240    LENGTH = 16, DOWN
+ 3   1   0      MAXIMUM DELAY
+ 4   0   0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 39000
SUBROUTINE EXECUTED AT  39000
INPUT:            OUTPUT:
```

```
HL= 40000            HL= 40000
PARAM+ 0   32        PARAM+ 0   32   ]
PARAM+ 1   0         PARAM+ 1   0    |
PARAM+ 2   240       PARAM+ 2   240  |-UNCHANGED
PARAM+ 3   0         PARAM+ 3   0    ]

NAME OF SUBROUTINE?
```

Notes

1. The program may "bomb" the system if the length of travel goes beyond video display memory boundaries.
2. The program may "bomb" the system if the x and y coordinates are improperly specified.
3. Use additional time-wasting instructions as required.
4. Delete time-wasting instructions as required. NOPs (all zeroes) may be substituted to shorten delay times.
5. Speed at maximum delay is about 85 character positions per second.

Program Listing

```
7F00                00100          ORG      7F00H              ;0522
                    00110 ;*****************************************************
                    00120 ;* MOVING DOT VERTICAL. MOVES DOT ALONG VERTICAL LINE   *
                    00130 ;* WITH VARYING TIME DELAY                               *
                    00140 ;*     INPUT: HL=> PARAMETER BLOCK                       *
                    00150 ;*            PARAM+0=STARTING CHAR POS'N (X)            *
                    00160 ;*            PARAM+1=STARTING LINE # (Y)                *
                    00170 ;*            PARAM+2=LENGTH OF TRAVEL IN CHAR POSNS     *
                    00180 ;*                    + IS UP, - IS DOWN                 *
                    00190 ;*            PARAM+3=TIME DELAY, 1=MIN 255/0=MAX        *
                    00200 ;*     OUTPUT:DOT MOVES ALONG VERTICAL LINE              *
                    00210 ;*****************************************************
                    00220 ;
7F00 F5             00230 MVVERT   PUSH     AF                 ;SAVE REGISTERS
7F01 C5             00240          PUSH     BC
7F02 D5             00250          PUSH     DE
7F03 E5             00260          PUSH     HL
7F04 DDE5           00270          PUSH     IX
7F06 FDE5           00280          PUSH     IY
7F08 CD7F0A         00290          CALL     0A7FH              ;***GET PB LOC'N***
7F0B E5             00300          PUSH     HL                 ;TRANSFER TO IX
7F0C DDE1           00310          POP      IX
7F0E 0606           00320          LD       B,6                ;ITERATION COUNT
7F10 DD6E01         00330          LD       L,(IX+1)           ;GET LINE #
7F13 2600           00340          LD       H,0                ;NOW IN HL
7F15 29             00350 MVV010   ADD      HL,HL                ;LINE# * 64
7F16 10FD           00360          DJNZ     MVV010               ;LOOP 'TIL DONE
7F18 01003C         00370          LD       BC,3C00H           ;START OF SCREEN
7F1B 09             00380          ADD      HL,BC              ;FIND LOC OF LINE START
7F1C DD4E00         00390          LD       C,(IX+0)           ;GET CHAR POSN (X)
7F1F 0600           00400          LD       B,0                ;NOW IN BC
7F21 09             00410          ADD      HL,BC              ;FIND ACTUAL LOC'N
7F22 DD4602         00420          LD       B,(IX+2)           ;GET LENGTH OF TRAVEL
7F25 CB78           00430          BIT      7,B                ;TEST SIGN
7F27 11C0FF         00440          LD       DE,-40H            ;INCREMENT FOR NEXT DOT
7F2A 2807           00450          JR       Z,MVV020           ;GO IF UP
7F2C 78             00460          LD       A,B                ;DOWN
7F2D ED44           00470          NEG                         ;FIND ABSOLUTE VALUE
7F2F 47             00480          LD       B,A                ;BACK TO B FOR DJNZ
```

```
7F30 114000    00490          LD      DE,40H          ;INCREMENT FOR DOWN
7F33 36BF      00500 MVV020   LD      (HL),0BFH         ;SET CHAR POS TO ALL ON
7F35 DD4E03    00510          LD      C,(IX+3)          ;GET DELAY COUNT
7F38 0D        00520 MVV030   DEC     C                   ;DECREMENT COUNT
7F39 FD2A0000  00530          LD      IY,(0)              ;WASTE TIME
7F3D FD2A0000  00540          LD      IY,(0)
7F41 FD2A0000  00550          LD      IY,(0)
7F45 FD2A0000  00560          LD      IY,(0)
7F49 20ED      00570          JR      NZ,MVV030           ;DELAY LOOP
7F4B 3680      00580          LD      (HL),80H          ;RESET CHAR POS
7F4D 19        00590          ADD     HL,DE             ;POINT TO NEXT POSITION
7F4E 10E3      00600          DJNZ    MVV020            ;LOOP FOR LENGTH OF LINE
7F50 FDE1      00610          POP     IY              ;RESTORE REGISTERS
7F52 DDE1      00620          POP     IX
7F54 E1        00630          POP     HL
7F55 D1        00640          POP     DE
7F56 C1        00650          POP     BC
7F57 F1        00660          POP     AF
7F58 C9        00670          RET                     ;RETURN TO CALLING PROG
0000           00680          END
00000 TOTAL ERRORS
```

MVVERT DECIMAL VALUES

```
245, 197, 213, 229, 221, 229, 253, 229, 205, 127,
10, 229, 221, 225, 6, 6, 221, 110, 1, 38,
0, 41, 16, 253, 1, 0, 60, 9, 221, 78,
0, 6, 0, 9, 221, 70, 2, 203, 120, 17,
192, 255, 40, 7, 120, 237, 68, 71, 17, 64,
0, 54, 191, 221, 78, 3, 13, 253, 42, 0,
0, 253, 42, 0, 0, 253, 42, 0, 0, 253,
42, 0, 0, 32, 237, 54, 128, 25, 16, 227,
253, 225, 221, 225, 225, 209, 193, 241, 201
```

CHKSUM= 81

NECDRV: NEC SPINWRITER DRIVER

System Configuration

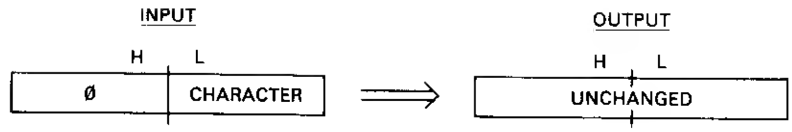
Model I.

Description

NECDRV is a printer driver for the serial NEC Spinwriter Printer or similar type of serial printer. Previous to use, the SETCOM subroutine must have been run to initialize the RS-232-C interface to the proper baud rate and other serial parameters. The NECDRV subroutine outputs a single character to the serial printer with automatic line feed. The wiring configuration for the Spinwriter cabling is shown in the figure below.

Input/Output Parameters

On input, the L register contains the character to be printed. On output the character has been printed and all registers are unchanged.



Algorithm

The NECDRV subroutine first gets the status from the RS-232-C controller holding register. If the transmitter holding register is not empty, the previous character has not been sent. If it is empty, the Clear to Send (CTS) line is checked. If there is a CTS, the character in HL is output. A test for a carriage return is then done. If the character is a carriage return, a line feed character is sent by a jump back to NEC010.

Sample Calling Sequence

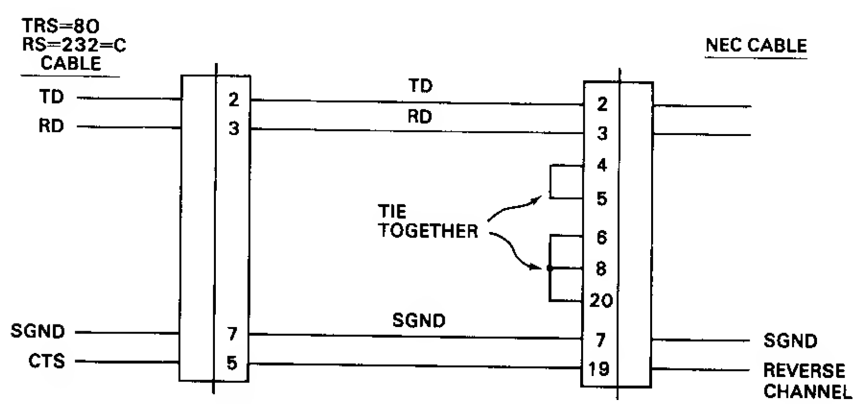
```
NAME OF SUBROUTINE? NECDRV
HL VALUE? 65   "A"
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 37000
SUBROUTINE EXECUTED AT  37000
INPUT:              OUTPUT:
HL= 65              HL= 65

NAME OF SUBROUTINE?
```

Notes

1. See the SETCOM subroutine for comments about setting up the RS-232-C interface.
2. Baud rates of 110 to 1200 may be used.

Program Listing



NEC spinwriter connections.

```
7F00            ORG     7F00H           ;0522
        ;**********************************************************
        ;* NEC SPINWRITER DRIVER. ROUTINE FOR USING NEC SPIN-     *
        ;* WRITER WITH SERIAL OUTPUT.                             *
        ;*    INPUT: HL=CHARACTER TO BE PRINTED                   *
        ;*    OUTPUT:CHARACTER PRINTED ON SPINWRITER              *
        ;**********************************************************
        ;
```

```
7F00 F5         00180 NECDRV  PUSH    AF              ;SAVE REGISTER
7F01 CD7F0A     00190         CALL    0A7FH           ;***GET CHARACTER***
7F04 3AEA00     00200 NEC010  LD      A,(0EAH)          ;GET STATUS
7F07 CB77       00210         BIT     6,A               ;TEST XMTR HOLDING REG
7F09 28F9       00220         JR      Z,NEC010          ;GO IF NOT EMPTY
7F0B DBE8       00230         IN      A,(0E8H)          ;GET CLEAR TO SEND
7F0D CB7F       00240         BIT     7,A               ;TEST
7F0F 28F3       00250         JR      Z,NEC010          ;GO IF NOT CTS
7F11 7D         00260         LD      A,L             ;PUT CHARACTER IN A
7F12 D3EB       00270         OUT     (0EBH),A        ;OUTPUT CHARACTER
7F14 FE0D       00280         CP      0DH             ;TEST FOR CR
7F16 2004       00290         JR      NZ,NEC090       ;GO IF NOT CR
7F18 3E0A       00300         LD      A,0AH           ;LINE FEED
7F1A 18E8       00310         JR      NEC010          ;OUTPUT LF
7F1C F1         00320 NEC090  POP     AF              ;RESTORE REGISTER
7F1D C9         00330         RET
0000            00340         END
00000 TOTAL ERRORS
```

```
NECDRV DECIMAL VALUES

245, 205, 127, 10, 58, 234, 0, 203, 119, 40,
249, 219, 232, 203, 127, 40, 243, 125, 211, 235,
254, 13, 32, 4, 62, 10, 24, 232, 241, 201

CHKSUM= 102
```

PRANDM: PSEUDO-RANDOM NUMBER GENERATOR

System Configuration

Model I, Model III, Model II Stand Alone.

Description

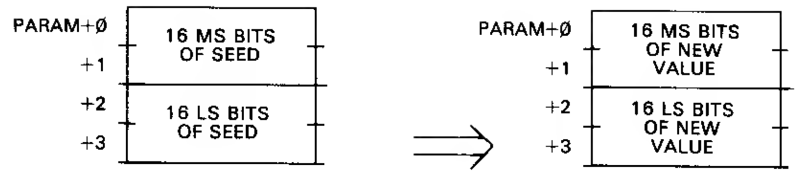
This subroutine returns a pseudo-random number in 32 bits. A pseudo-random number differs from a random number in that it is repeatable. If the same "seed" value is used, the same sequence of numbers as previously generated will be repeated. At the same time, the sequence of numbers will appear to be randomly distributed and can be utilized as random numbers for games, simulations, and modeling.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The four bytes of the parameter block contain the seed, or starting value, of the pseudo-random number sequence. The seed value may not be zero.

On output, the four bytes of the parameter block contain the next pseudo-random number in sequence.

INPUT		OUTPUT
H L		H L
POINTER TO PARAM+0	⟹	UNCHANGED



Algorithm

A pseudo-random number sequence with a relatively long cycle time can be generated by multiplying a 32-bit value by an odd power of 5. In this case, the third power of five is used to multiply the seed value by 125.

The 32-bit seed is picked up from the parameter block and put into DE, HL. DE, HL is now added to itself three times in the PRA010 loop to multiply the original seed by 128. Next, the original seed value is put into BC. BC is then subtracted from DE, HL three times to produce a result that is the original number times 125. This value is then stored back into the parameter block to be used as the new seed.

Sample Calling Sequence

```
NAME OF SUBROUTINE? PRANDM
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0  2  1  ]
+ 2  2  1  ]- SEED = 00010001H
+ 4  0  0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 37000
SUBROUTINE EXECUTED AT  37000
INPUT:               OUTPUT:
HL= 40000            HL= 40000
PARAM+ 0  1          PARAM+ 0  125 ]
PARAM+ 1  0          PARAM+ 1  0   ]- NEW VALUE = 007D007DH
PARAM+ 2  1          PARAM+ 2  125 ]
PARAM+ 3  0          PARAM+ 3  0   ]

NAME OF SUBROUTINE?
```

Notes

1. Initialize the seed value at the beginning of the sequence with a nonzero value. Thereafter, simply call PRANDM with the previous pseudo-random number in the parameter block.

2. An initial seed of an odd number generates all odd numbers, an initial seed of an even number, even numbers. You may use only the most significant n bits of the 32 bits to obtain odd and even numbers.

Program Listing

```
7F00                  ORG      7F00H             ;0522
          ;*****************************************************
          ;* PSEUDO-RANDOM NUMBER ROUTINE. GENERATES A PSEUDO-   *
          ;* RANDOM (REPEATABLE) NUMBER.                         *
          ;*    INPUT: HL=> PARAMETER BLOCK                      *
          ;*             PARAM+0,+1=16 MS BITS OF SEED           *
          ;*             PARAM+2,+3=16 LS BITS OF SEED           *
          ;*    OUTPUT:PARAM+0,+1=16 MS BITS OF NEW VALUE        *
          ;*             PARAM+2,+3=16 LS BITS OF NEW VALUE      *
          ;*****************************************************
```

```
                 00200 ;
7F00 F5          00210 PRANDM  PUSH    AF              ;SAVE REGISTERS
7F01 C5          00220         PUSH    BC
7F02 D5          00230         PUSH    DE
7F03 E5          00240         PUSH    HL
7F04 DDE5        00250         PUSH    IX
7F06 CD7F0A      00260         CALL    0A7FH           ;***GET PAR BL ADDR***
7F09 E5          00270         PUSH    HL              ;TRANSFER TO IX
7F0A DDE1        00280         POP     IX
7F0C DD5E00      00290         LD      E,(IX+0)        ;DE HOLDS MS SEED
7F0F DD5601      00300         LD      D,(IX+1)
7F12 DD6E02      00310         LD      L,(IX+2)        ;HL HOLDS LS SEED
7F15 DD6603      00320         LD      H,(IX+3)
7F18 0607        00330         LD      B,7             ;FOR LOOP COUNT
7F1A 29          00340 PRA010  ADD     HL,HL             ;2 TIMES LS 16 BITS
7F1B EB          00350         EX      DE,HL             ;MS NOW IN HL
7F1C ED6A        00360         ADC     HL,HL             ;2 TIME MS 16 BITS
7F1E EB          00370         EX      DE,HL
7F1F 10F9        00380         DJNZ    PRA010            ;7 TIMES=TIMES 128
7F21 3E03        00390         LD      A,3             ;COUNT FOR SUBTRACT
7F23 DD4E02      00400 PRA020  LD      C,(IX+2)          ;GET LS 16 BITS OF SEED
7F26 DD4603      00410         LD      B,(IX+3)
7F29 B7          00420         OR      A                 ;RESET CARRY
7F2A ED42        00430         SBC     HL,BC             ;SUBTRACT
7F2C EB          00440         EX      DE,HL             ;SWAP
7F2D DD4E00      00450         LD      C,(IX+0)          ;GET MS 16 BITS OF SEED
7F30 DD4601      00460         LD      B,(IX+1)
7F33 ED42        00470         SBC     HL,BC             ;SUBTRACT
7F35 EB          00480         EX      DE,HL             ;SWAP BACK
7F36 3D          00490         DEC     A                 ;3 TIMES=SEED*125
7F37 20EA        00500         JR      NZ,PRA020         ;GO IF NOT 3
7F39 DD7300      00510         LD      (IX+0),E        ;STORE NEW VALUE
7F3C DD7201      00520         LD      (IX+1),D
7F3F DD7502      00530         LD      (IX+2),L
7F42 DD7403      00540         LD      (IX+3),H
7F45 DDE1        00550         POP     IX              ;RESTORE REGISTERS
7F47 E1          00560         POP     HL
7F48 D1          00570         POP     DE
7F49 C1          00580         POP     BC
7F4A F1          00590         POP     AF
7F4B C9          00600         RET                     ;RETURN
0000             00610         END
00000 TOTAL ERRORS
```

```
PRANDM DECIMAL VALUES

 245, 197, 213, 229, 221, 229, 205, 127, 10, 229,
 221, 225, 221, 94, 0, 221, 86, 1, 221, 110,
 2, 221, 102, 3, 6, 7, 41, 235, 237, 106,
 235, 16, 249, 62, 3, 221, 78, 2, 221, 70,
 3, 183, 237, 66, 235, 221, 78, 0, 221, 70,
 1, 237, 66, 235, 61, 32, 234, 221, 115, 0,
 221, 114, 1, 221, 117, 2, 221, 116, 3, 221,
 225, 225, 209, 193, 241, 201

CHKSUM= 229
```

RANDOM: RANDOM NUMBER GENERATOR

System Configuration

Model I, Model III, Model II Stand Alone.

Description

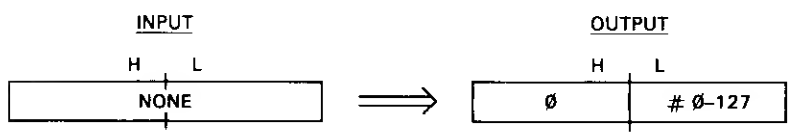
This subroutine returns a true random number of 0 through 127, provided certain conditions are met. If the subroutine is called at unpredictable intervals the number returned will be truly random. An example of this would be a CALL to RANDOM after a keypress from the TRS-80 keyboard. If RANDOM is called repetitively to generate 100 "random" numbers, however, the numbers generated will not be random. It's very possible in this case that the number of microprocessor cycles between each CALL will be fixed, and that the resulting numbers will simply differ by a fixed amount.

RANDOM generates random numbers by using the count in the R register. As R is used for refresh and is continually counting from 0 through 127, the event that causes the CALL to random must be "asynchronous" compared to the Z-80 timing and must occur over relatively long periods of time (hundreths of seconds). RANDOM is simply a means to use the asynchronous event to conveniently generate a number from 0 through 127.

Input/Output Parameters

There are no input parameters to RANDOM.

On output, RANDOM returns the count in the R register in HL. H will be 0 and L will be a value of 0 through 127.



Algorithm

Obtaining the count from the R register can be compared to spinning a wheel that has 128 divisions numbered 0 through 127. The wheel is stopped at random times to yield a true random number.

R is incremented from 0 through 127 to provide a refresh address for the TRS-80 dynamic RAM. An increment occurs each "fetch" cycle of an instruction, which is either once or twice per instruction (some instructions have two fetch or M1 cycles). If a typical instruction takes 5 microseconds, R counts 200,000 times per second, making the time between external events such as keypresses sufficiently large to generate true random numbers.

Sample Calling Sequence

```
NAME OF SUBROUTINE? RANDOM
HL VALUE? Ø
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 38000
SUBROUTINE EXECUTED AT  38000
INPUT:              OUTPUT:
HL= Ø               HL= 16   RANDOM #

NAME OF SUBROUTINE?
```

Notes

1. To get a number in a range other than 0–127, subtract the range required from the value in HL until the number is less than the range required. If the number returned is 99, for example, and the number required is 0–9, then subtracting 10 until the result is less than 10 produces 9, a number in the range required.

Program Listing

```
7F00            00100           ORG     7F00H           ;0520
                00110 ;**********************************************************
                00120 ;* RANDOM NUMBER GENERATOR. GENERATES A TRUE RANDOM NUM-*
                00130 ;* BER PROVIDED CALLED AT ASYNCHRONOUS TIMES!            *
                00140 ;*    INPUT: NONE                                         *
                00150 ;*    OUTPUT:RANDOM NUMBER 0-127 IN HL                    *
                00160 ;**********************************************************
                00170 ;
7F00 F5         00180 RANDOM    PUSH    AF              ;SAVE REGISTER
7F01 ED5F       00190           LD      A,R             ;GET 0-127 FROM R
7F03 6F         00200           LD      L,A             ;NOW IN L
7F04 2600       00210           LD      H,0             ;NOW IN HL
7F06 F1         00220           POP     AF              ;RESTORE REGISTER
7F07 C39A0A     00230           JP      0A9AH           ;***RETURN WITH ARG***
7F0A C9         00240           RET                     ;NON-BASIC RETURN
0000            00250           END
00000 TOTAL ERRORS
```

```
RANDOM DECIMAL VALUES


 245, 237, 95, 111, 38, 0, 241, 195, 154, 10,
 201

CHKSUM= 247
```

RCRECD: READ CASSETTE RECORD

System Configuration

Model I, Model III.

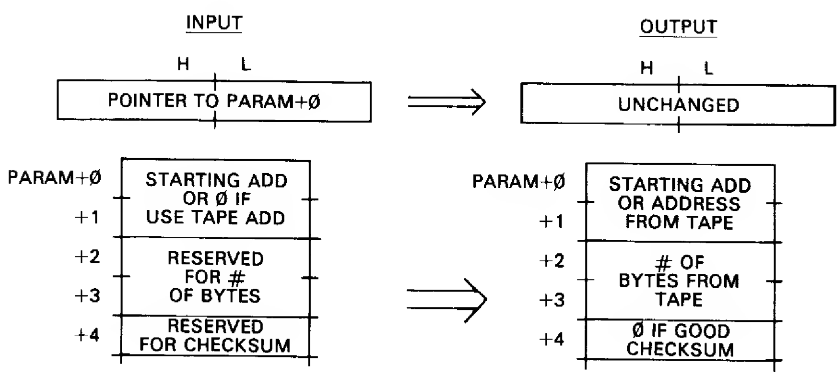
Description

RCRECD reads a previously written record from cassette to memory. The WCRECD subroutine must have been used to generate the cassette record. The record may be any number of bytes, from 1 to the limits of memory. The record is prefixed by a four-byte header that holds the starting address and number of bytes in the remainder of the record. The record is terminated by a checksum byte that is the additive checksum of all bytes in the record. Data in the record may represent any type of data the user desires; the record is read in as a "core image."

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block are the starting address of the data to be read in, in standard Z-80 address format, least significant byte followed by most significant byte. If the starting address of the cassette record header is to be used, this parameter is 0. The next two bytes of the parameter block are reserved for the number of bytes value from the record header. The next byte is reserved for the checksum from the record header.

On output, the contents of the parameter block is unchanged and the record has been read from cassette. PARAM+2,+3 contain the starting address of the data from tape, if this address was to be used. PARAM+4 contains the checksum for the read operation. If this value is a zero, the tape data has been read correctly; otherwise, an invalid read of one or more cassette bytes has occurred.



Algorithm

The RCRECD subroutine uses Level II or Level III ROM subroutines to perform the write. First, a CALL is made to 212H to select cassette 0. Next, a call is made to 296H to bypass the leader and sync byte on the cassette.

The four-byte header is next read from the cassette record. The number of bytes from the cassette record is saved in the parameter block. The starting address from the cassette record is saved if the starting address was zero. At this time also, the B register contains the checksum of the first four cassette bytes.

The value from PARAM+0, +1 (original starting address or starting address from cassette) is picked up at RCR020. The code from RCR030 on is a loop to read a cassette byte by a CALL to 235H, store the byte in memory via the HL pointer, increment the pointer and decrement the byte count, and checksum each byte. When DE has been decremented down to zero, the read of the body of the cassette record is done, and a final read is performed to pick up the checksum byte from the cassette.

The checksum value in B is subtracted from the cassette checksum, and the result stored in the parameter block. The two should be equal, resulting in a difference of zero. Finally, a CALL to 1F8H is done to deselect the cassette.

Sample Calling Sequence

```
NAME OF SUBROUTINE? RCRECD
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0  2  0   USE TAPE ADDRESS
+ 2  2  0 ]
+ 4  1  0 ]-INITIALIZE FOR EXAMPLE
+ 5  0  0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 37000
SUBROUTINE EXECUTED AT  37000
INPUT:                OUTPUT:
HL= 40000             HL= 40000
PARAM+ 0  0           PARAM+ 0  0  ]-ADDRESS FROM TAPE (3C00H)
PARAM+ 1  0           PARAM+ 1  60 ]
PARAM+ 2  0           PARAM+ 2  0  ]-1024 BYTES
PARAM+ 3  0           PARAM+ 3  4  ]
PARAM+ 4  0           PARAM+ 4  0    CHECKSUM OK

NAME OF SUBROUTINE?
```

Notes

1. This subroutine uses cassette 0 only.
2. For 500 baud tape operations, each 1000 bytes will take about 20 seconds.
3. This subroutine does not save registers.

Program Listing

```
7F00                     ORG     7F00H           ;0520
                ;*****************************************************
                ;* READ RECORD FROM CASSETTE. READS RECORD PREVIOUSLY  *
                ;* WRITTEN BY WCRECD ROUTINE.                          *
                ;*    INPUT: HL=> PARAMETER BLOCK                      *
                ;*           PARAM+0,+1=STRTNG ADDR OR 0 IF TAPE ADDRS *
                ;*           PARAM+2,+3=RESERVED FOR NUMBER OF BYTES   *
                ;*           PARAM+4=RESERVED FOR CHECKSUM             *
                ;*    OUTPUT:PARAM+0,+1=STARTING ADDRESS, ORIG OR TAPE *
                ;*           PARAM+2,+3=# OF BYTES FROM TAPE RECORD    *
                ;*           PARAM+4=CHECKSUM. 0 IF VALID, ELSE NON-ZER *
                ;*****************************************************
                ;
7F00 F3         RCRECD   DI                      ;DISABLE INTERRUPTS
7F01 AF                  XOR     A               ;ZERO A
7F02 CD1202              CALL    212H            ;SELECT CASSETTE 0
7F05 CD9602              CALL    296H            ;BYPASS LEADER
7F08 CD7F0A              CALL    0A7FH           ;***GET PB LOC'N***
7F0B E5                  PUSH    HL              ;TRANSFER TO IX
7F0C DDE1                POP     IX
7F0E DDE5                PUSH    IX              ;SAVE
7F10 CD3502              CALL    235H            ;GET START LSB
7F13 6F                  LD      L,A             ;SAVE
7F14 E5                  PUSH    HL
7F15 CD3502              CALL    235H            ;GET START MSB
7F18 E1                  POP     HL              ;RESTORE LSB
7F19 67                  LD      H,A             ;MERGE MSB
7F1A E5                  PUSH    HL
7F1B CD3502              CALL    235H            ;GET # LSB
7F1E 5F                  LD      E,A             ;SAVE
7F1F D5                  PUSH    DE
```

```
7F20 CD3502    00410          CALL   235H        ;GET # MSB
7F23 D1        00420          POP    DE          ;RESTORE #
7F24 57        00430          LD     D,A
7F25 E1        00440          POP    HL          ;RESTORE STARTING ADDRESS
7F26 DDE1      00450          POP    IX          ;POINTER TO PAR BLOCK
7F28 7A        00460          LD     A,D         ;INITIALIZE CHECKSUM
7F29 83        00470          ADD    A,E
7F2A 84        00480          ADD    A,H
7F2B 85        00490          ADD    A,L
7F2C 47        00500          LD     B,A         ;SAVE CHECKSUM
7F2D DD7302    00510          LD     (IX+2),E    ;SAVE # OF BYTES
7F30 DD7203    00520          LD     (IX+3),D
7F33 DD7E00    00530          LD     A,(IX+0)    ;GET STARTING ADDRESS
7F36 B7        00540          OR     A           ;TEST FOR 0
7F37 2006      00550          JR     NZ,RCR020   ;GO IF USE ADDRESS IN PB
7F39 DD7500    00560          LD     (IX+0),L    ;STORE TAPE ADDRESS
7F3C DD7401    00570          LD     (IX+1),H
7F3F DD6E00    00580 RCR020   LD     L,(IX+0)    ;GET STARTING ADDRESS
7F42 DD6601    00590          LD     H,(IX+1)
7F45 DDE5      00600          PUSH   IX          ;SAVE POINTER
7F47 C5        00610 RCR030   PUSH   BC            ;SAVE CHECKSUM
7F48 D5        00620          PUSH   DE            ;SAVE ENDING ADDRESS
7F49 E5        00630          PUSH   HL            ;SAVE CURRENT LOCATION
7F4A CD3502    00640          CALL   235H          ;READ NEXT BYTE
7F4D E1        00650          POP    HL            ;RESTORE POINTER
7F4E D1        00660          POP    DE            ;RESTORE ENDING LOC'N
7F4F C1        00670          POP    BC            ;RESTORE CHECKSUM
7F50 77        00680          LD     (HL),A        ;STORE BYTE
7F51 80        00690          ADD    A,B           ;ADD IN CHECKSUM
7F52 47        00700          LD     B,A           ;SAVE CHECKSUM
7F53 23        00710          INC    HL            ;BUMP POINTER
7F54 1B        00720          DEC    DE            ;DECREMENT # OF BYTES
7F55 7A        00730          LD     A,D           ;TEST FOR 0
7F56 B3        00740          OR     E
7F57 20EE      00750          JR     NZ,RCR030     ;GO IF NOT LAST BYTE
7F59 C5        00760          PUSH   BC          ;SAVE CHECKSUM
7F5A CD3502    00770          CALL   235H        ;READ CHECKSUM BYTE
7F5D C1        00780          POP    BC          ;RESTORE CHECKSUM
7F5E DDE1      00790          POP    IX          ;RESTORE POINTER
7F60 90        00800          SUB    B           ;TEST CHECKSUM
7F61 DD7704    00810          LD     (IX+4),A    ;STORE FLAG
7F64 CDF801    00820          CALL   1F8H        ;DESELECT
7F67 C9        00830          RET                ;RETURN TO CALLING PROG
0000           00840          END
00000 TOTAL ERRORS
```

RCRECD DECIMAL VALUES

```
243, 175, 205, 18, 2, 205, 150, 2, 205, 127,
10, 229, 221, 225, 221, 229, 205, 53, 2, 111,
229, 205, 53, 2, 225, 103, 229, 205, 53, 2,
95, 213, 205, 53, 2, 209, 87, 225, 221, 225,
122, 131, 132, 133, 71, 221, 115, 2, 221, 114,
3, 221, 126, 0, 183, 32, 6, 221, 117, 0,
221, 116, 1, 221, 110, 0, 221, 102, 1, 221,
229, 197, 213, 229, 205, 53, 2, 225, 209, 193,
119, 128, 71, 35, 27, 122, 179, 32, 238, 197,
205, 53, 2, 193, 221, 225, 144, 221, 119, 4,
205, 248, 1, 201
```

CHKSUM= 185

RDCOMS: READ RS-232-C SWITCHES

System Configuration

Model I.

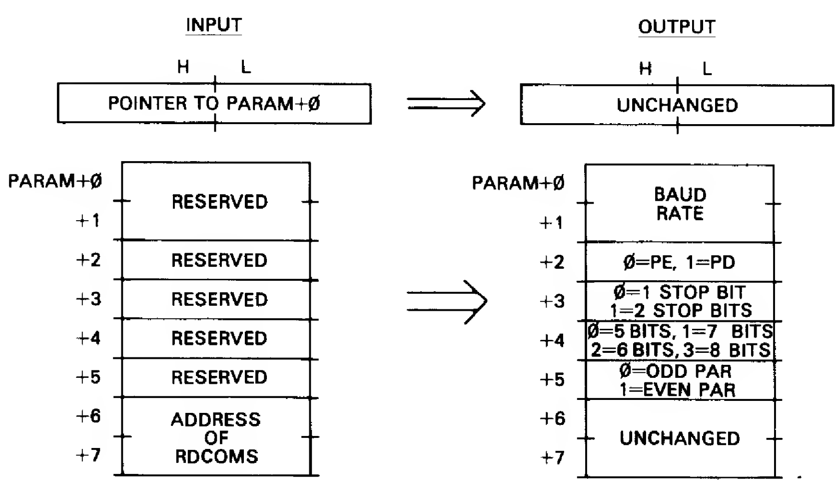
Description

RDCOMS reads the configuration of switches on the RS-232-C controller board. The configuration of the switches is analyzed and put into separate parameters. RDCOMS may be used to verify that the switches are set correctly without having to reopen the RS-232-C access and reset the switches.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first six bytes of the parameter block are reserved for the results of the read. The last two bytes of the parameter block (PARAM+6,+7) hold the address of RDCOMS in standard Z-80 address format, least significant byte followed by most significant byte. This address can be obtained from the USR call address in BASIC or in the assembly-language CALL address.

On output, the first two bytes of the parameter block contain the baud rate for which the RS-232-C interface is set, 110, 150, 300, 600, 1200, 2400, 4800, or 9600. The next byte is set to a zero if parity is enabled, or to a one if parity is disabled. The next byte of the parameter block is set to a zero if one stop bit is used, or to a one if two stop bits are used. The next byte contains the number of bits in the RS-232-C transfer; 0 is 5 bits, 1 is 7 bits, 2 is 6 bits, or 3 is 8 bits. The next byte contains a zero if odd parity is used, or a one if even parity is used.



Algorithm

The SETCOM subroutine reads the switches and strips and aligns the fields into the proper format for the parameter block.

First the switches are read by an "IN A,(0E9H)." Next, the parity type is obtained by a rotate left and an AND of 1 and stored in the parameter block. The switch byte is then rotated again two bits and an AND of 3 picks up the number of bits, which is stored in the parameter block. The switch byte is then rotated left and an AND of 1 picks up the number of stop bits, which is stored in the parameter block. The switch byte is then rotated left and an AND of 1 picks up the parity enable/disable bit, which is stored in the parameter block. The switch byte is then rotated left three times. An AND of 7 obtains the baud rate index.

The baud rate index is put into HL and an ADD of HL to itself is done to multiply the index by two. The result is added to the location of RDCOMS and to the displacement of TABBD. HL now points to the TABBD entry, which is the baud rate corresponding to the switch code. This code is picked up from the table and stored in the parameter block.

Sample Calling Sequence

```
NAME OF SUBROUTINE? RDCOMS
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0  2  0
+ 2  2  0        INITIALIZE FOR EXAMPLE
+ 4  2  0
+ 6  2  37890    START OF RDCOMS
+ 8  0  0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 37890
SUBROUTINE EXECUTED AT  37890
INPUT:              OUTPUT:
HL= 40000           HL= 40000
PARAM+ 0  0         PARAM+ 0  176   1200 BAUD
PARAM+ 1  0         PARAM+ 1  4
PARAM+ 2  0         PARAM+ 2  0     PE
PARAM+ 3  0         PARAM+ 3  1     TWO STOP BITS
PARAM+ 4  0         PARAM+ 4  2     SIX BIT LENGTH
PARAM+ 5  0         PARAM+ 5  1     EVEN PARITY
PARAM+ 6  2         PARAM+ 6  2     UNCHANGED
PARAM+ 7  148       PARAM+ 7  148

NAME OF SUBROUTINE?
```

Notes

1. Note transposed order of number of bits.

```
7F00                    ORG     7F00H           ;0522
                ;*****************************************************
                ;* READ RS-232-C SWITCHES. READS THE RS-232-C BOARD    *
                ;* SWITCHES.                                            *
                ;*     INPUT: HL=> PARAMETER BLOCK                      *
                ;*            PARAM+0 - PARAM+5: SEE OUTPUT             *
                ;*            PARAM+6,+7: ADDRESS OF RDCOMS             *
                ;*     OUTPUT:HL=> PARAMETER BLOCK                      *
                ;*            PARAM+0,+1=BAUD RATE - 110, 150, 300, 600, *
                ;*                      120, 2400, 4800, 9600           *
                ;*            PARAM+2=0=PARITY ENABLED, 1=PARITY DISAB  *
                ;*            PARAM+3=0=ONE STOP BIT, 1=TWO STOP BITS   *
                ;*            PARAM+4=0=5 BITS, 1=7 BITS, 2=6 BITS, 3=8 *
                ;*                    BITS                              *
                ;*            PARAM+5=0=ODD PARITY, 1=EVEN              *
                ;*****************************************************
                ;
7F00 F5         RDCOMS  PUSH    AF              ;SAVE REGISTERS
7F01 C5                 PUSH    BC
7F02 D5                 PUSH    DE
7F03 E5                 PUSH    HL
7F04 DDE5               PUSH    IX
7F06 CD7F0A             CALL    0A7FH           ;***GET PB LOC'N***
7F09 E5                 PUSH    HL              ;TRANSFER TO IX
7F0A DDE1               POP     IX
7F0C DBE9               IN      A,(0E9H)        ;READ SWITCHES
7F0E 47                 LD      B,A             ;SAVE IN B
7F0F CB00               RLC     B               ;ALIGN
7F11 78                 LD      A,B
7F12 E601               AND     1               ;GET PARITY TYPE
7F14 DD7705             LD      (IX+5),A        ;STORE
7F17 CB00               RLC     B               ;ALIGN
7F19 CB00               RLC     B
7F1B 78                 LD      A,B
7F1C E603               AND     3               ;GET # OF BITS
7F1E DD7704             LD      (IX+4),A        ;STORE
7F21 CB00               RLC     B               ;ALIGN
7F23 78                 LD      A,B
7F24 E601               AND     1               ;GET # OF STOP BITS
7F26 DD7703             LD      (IX+3),A        ;STORE
7F29 CB00               RLC     B               ;ALIGN
7F2B 78                 LD      A,B
7F2C E601               AND     1               ;GET PARITY ENAB/DIS
7F2E DD7702             LD      (IX+2),A        ;STORE
7F31 CB00               RLC     B               ;ALIGN
7F33 CB00               RLC     B
7F35 CB00               RLC     B
7F37 78                 LD      A,B
7F38 E607               AND     7               ;GET BAUD INDEX
7F3A 6F                 LD      L,A             ;BAUD INDEX NOW IN L
7F3B 2600               LD      H,0             ;NOW IN HL
7F3D 29                 ADD     HL,HL           ;INDEX*2
7F3E DD5E06             LD      E,(IX+6)        ;LOCATION OF RDCOMS
7F41 DD5607             LD      D,(IX+7)
7F44 19                 ADD     HL,DE           ;INDEX PLUS BASE ADDRESS
7F45 115900             LD      DE,TABBD        ;BAUD RATE TABLE
7F48 19                 ADD     HL,DE           ;INDEX + BASE + TABLE DIS
7F49 7E                 LD      A,(HL)          ;GET TABLE ENTRY
7F4A DD7700             LD      (IX+0),A        ;STORE
7F4D 23                 INC     HL              ;POINT TO NEXT BYTE
7F4E 7E                 LD      A,(HL)          ;GET NEXT BYTE
7F4F DD7701             LD      (IX+1),A        ;STORE
```

```
7F52 DDE1      00720          POP     IX              ;RESTORE REGISTERS
7F54 E1        00730          POP     HL
7F55 D1        00740          POP     DE
7F56 C1        00750          POP     BC
7F57 F1        00760          POP     AF
7F58 C9        00770          RET                     ;RETURN TO CALLING PROG
0059           00780 TABBD    EQU     $-RDCOMS        ;BAUD RATE TABLE
7F59 6E00      00790          DEFW    110
7F5B 9600      00800          DEFW    150
7F5D 2C01      00810          DEFW    300
7F5F 5802      00820          DEFW    600
7F61 B004      00830          DEFW    1200
7F63 6009      00840          DEFW    2400
7F65 C012      00850          DEFW    4800
7F67 8025      00860          DEFW    9600
0000           00870          END
00000 TOTAL ERRORS
```

```
RDCOMS DECIMAL VALUES

245, 197, 213, 229, 221, 229, 205, 127, 10, 229,
221, 225, 219, 233, 71, 203, 0, 120, 230, 1,
221, 119, 5, 203, 0, 203, 0, 120, 230, 3,
221, 119, 4, 203, 0, 120, 230, 1, 221, 119,
3, 203, 0, 120, 230, 1, 221, 119, 2, 203,
0, 203, 0, 203, 0, 120, 230, 7, 111, 38,
0, 41, 221, 94, 6, 221, 86, 7, 25, 17,
89, 0, 25, 126, 221, 119, 0, 35, 126, 221,
119, 1, 221, 225, 225, 209, 193, 241, 201, 110,
0, 150, 0, 44, 1, 88, 2, 176, 4, 96,
9, 192, 18, 128, 37

CHKSUM= 122
```

READDS: READ DISK SECTOR

System Configuration

Model I.

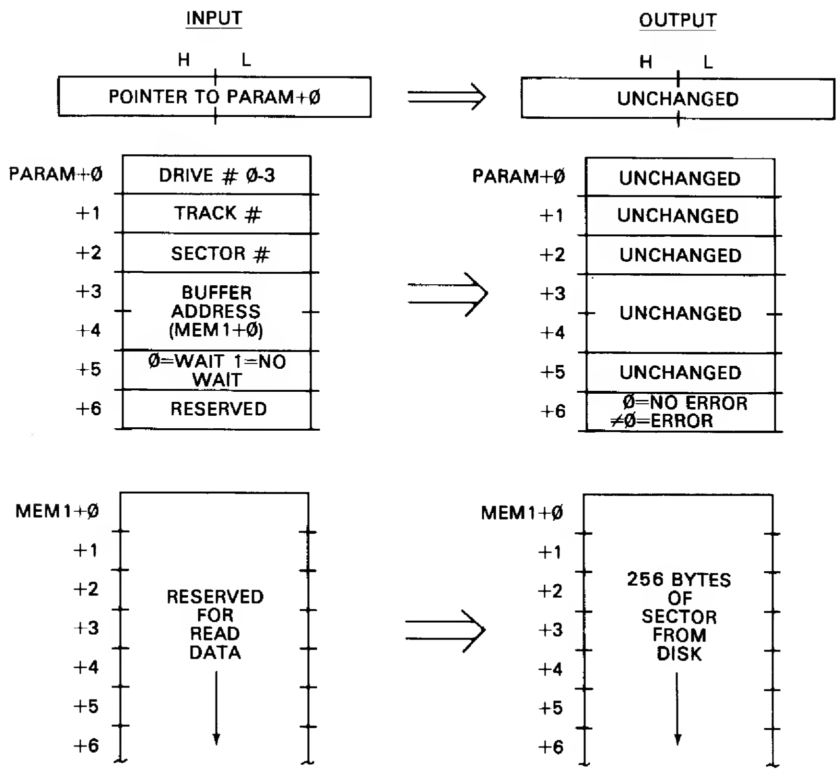
Description

READDS reads one sector from a specified disk drive into a 256-byte user buffer. The user must know where a particular file is and what sectors are involved to utilize this subroutine; it is not a general-purpose "file manage" subroutine.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first byte of the parameter block contains the disk drive number, 0 to 3, corresponding to disk drives 1 through 4. The next byte of the parameter block contains the track number, 0 through N. (The standard TRS-80 uses disk drives with 35 tracks; other drives are available for 40 tracks.) The next byte is the sector number, 0 through N (0 through 9 will be the most common range). The next two bytes are the user buffer area for the read in standard Z-80 address format, least significant byte followed by most significant byte. The next byte contains a zero if a wait is to occur until the disk drive motor is brought up to speed; the byte contains a 1 if the motor is running (disk operation has just been completed) and no wait is necessary. The next byte (PARAM+6) is reserved for the status of the disk read on output.

On output, all parameters remain unchanged except for PARAM+6, which contains the status of the read. Status is 0 for a successful read, or nonzero if an error occurred during any portion of the read. If an error did not occur, the specified disk sector has been read into the buffer area.



Algorithm

The disk drive number in L is first converted to the proper select configuration at REA010. The select byte is then output to disk memory-mapped address 37E0H to select one of the disk drives.

The wait bit is then examined. If this bit is a zero, the loop at REA015 counts HL through 65,536 counts to wait until the disk drive motor is up to speed before continuing.

The disk status is then examined (REA020). If the disk is not busy, the track number is loaded into the disk controller track register (37EFH) and a seek command is given (37ECH) to cause the controller to "seek" the track for the operation. A series of time-wasting instructions is then done.

The code at REA030 gets the disk status after completion of the seek and ANDs it with a "proper result" mask. If the status is normal, the read continues, otherwise an "abnormal" completion is done to REA090.

The sector address from the parameter block is next output to the controller sector register (37EEH). Two time-wasting instructions are then done.

A read command is then isued to the disk controller command register (37ECH). Further time-wasting instructions are done.

The loop at REA040 performs the actual read of the disk sector. A total of 256 separate reads is done. HL contains the disk address of 37ECH, DE contains a pointer to the buffer address, and BC contains the data register address of the disk controller. For each of the 256 reads, status is checked. If bit 0 is set, all 256 bytes have been read. If bit 1 of the status is set, the disk controller is still busy and a loop back to REA040 is done. If bit 1 of the status is not set the next byte is read, stored in memory, and the memory buffer pointer incremented.

At the automatic (by the controller) termination of the read, status is again read, and an AND of 1CH is done to check for the proper completion bits. The status is stored back into the parameter block.

Sample Calling Sequence

```
NAME OF SUBROUTINE? READDS
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0  1  0       DRIVE 0
+ 1  1  17      TRACK 17
+ 2  1  0       SECTOR 0
+ 3  2  45000   BUFFER
+ 5  1  0       WAIT
+ 6  1  0
+ 7  0  0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 38000
SUBROUTINE EXECUTED AT  38000
INPUT:               OUTPUT:
HL= 40000            HL= 40000
PARAM+ 0  0          PARAM+ 0  0    ┐
PARAM+ 1  17         PARAM+ 1  17   │
PARAM+ 2  0          PARAM+ 2  0    ├ UNCHANGED
PARAM+ 3  200        PARAM+ 3  200  │
PARAM+ 4  175        PARAM+ 4  175  │
PARAM+ 5  0          PARAM+ 5  0    ┘
PARAM+ 6  0          PARAM+ 6  0     STATUS = OK

NAME OF SUBROUTINE?
```

Notes

1. Always perform an RESTDS operation before doing initial disk I/O to reset the disk controller.

```
7F00               00100            ORG     7F00H           ;0522
                   00110 ;*****************************************************
                   00120 ;* READ DISK SECTOR. READS SPECIFIED TRACK, SECTOR INTO *
                   00130 ;* MEMORY BUFFER.                                      *
                   00140 ;*     INPUT: HL=> PARAMETER BLOCK                     *
                   00150 ;*            PARAM+0=DRIVE #, 0 - 3                   *
                   00160 ;*            PARAM+1=TRACK #, 0 - N                   *
                   00170 ;*            PARAM+2=SECTOR #, 0 - N                  *
                   00180 ;*            PARAM+3,+4=BUFFER ADDRESS                *
                   00190 ;*            PARAM+5=0=WAIT AFTER SELECT, 1=NO WAIT   *
                   00200 ;*            PARAM+6=RESERVED FOR STATUS              *
                   00210 ;*    OUTPUT:TRACK, SECTOR READ INTO BUFFER            *
                   00220 ;*            PARAM+6=STATUS, 0=OK, 1=BAD              *
                   00230 ;*****************************************************
                   00240 ;
7F00 F5            00250 READDS    PUSH    AF              ;SAVE REGISTERS
7F01 C5            00260           PUSH    BC
7F02 D5            00270           PUSH    DE
7F03 E5            00280           PUSH    HL
7F04 DDE5          00290           PUSH    IX
7F06 CD7F0A        00300           CALL    0A7FH           ;***GET PB LOC'N***
7F09 E5            00310           PUSH    HL              ;TRANSFER TO IX
7F0A DDE1          00320           POP     IX
7F0C DD7E00        00330           LD      A,(IX+0)        ;GET DRIVE #
7F0F 3C            00340           INC     A               ;INCREMENT BY ONE
7F10 47            00350           LD      B,A             ;PUT IN B FOR CONVERT
7F11 3E80          00360           LD      A,80H           ;MASK
7F13 07            00370 REA010    RLCA                     ;ALIGN FOR SELECT
7F14 10FD          00380           DJNZ    REA010           ;CONVERT TO ADDRESS
7F16 32E037        00390           LD      (37E0H),A       ;SELECT DRIVE
7F19 DD7E05        00400           LD      A,(IX+5)        ;GET WAIT/NO WAIT
7F1C B7            00410           OR      A               ;TEST
7F1D 2008          00420           JR      NZ,REA020       ;GO IF NO WAIT
7F1F 210000        00430           LD      HL,0            ;WAIT COUNT
7F22 2B            00440 REA015    DEC     HL               ;DELAY LOOP 6
7F23 7D            00450           LD      A,L              ;TEST DONE 4
7F24 B4            00460           OR      H                ;4
7F25 20FB          00470           JR      NZ,REA015        ;LOOP UNTIL HL=0 7/12
7F27 3AEC37        00480 REA020    LD      A,(37ECH)        ;GET STATUS
7F2A CB47          00490           BIT     0,A              ;TEST BUSY
7F2C 20F9          00500           JR      NZ,REA020        ;LOOP IF BUSY
7F2E DD7E01        00510           LD      A,(IX+1)        ;GET TRACK NUMBER
7F31 32EF37        00520           LD      (37EFH),A       ;OUTPUT TRACK #
7F34 C5            00530           PUSH    BC              ;WASTE TIME
7F35 C1            00540           POP     BC
7F36 3E17          00550           LD      A,17H           ;SEEK COMMAND
7F38 32EC37        00560           LD      (37ECH),A       ;OUTPUT
7F3B C5            00570           PUSH    BC              ;WASTE TIME
7F3C C1            00580           POP     BC
7F3D C5            00590           PUSH    BC
7F3E C1            00600           POP     BC
7F3F 3AEC37        00610 REA030    LD      A,(37ECH)         ;GET STATUS
7F42 CB47          00620           BIT     0,A               ;TEST BUSY
7F44 20F9          00630           JR      NZ,REA030         ;LOOP IF BUSY
7F46 E698          00640           AND     98H             ;TEST FOR NORMAL COMPL
7F48 202C          00650           JR      NZ,REA090       ;GO IF ABNORMAL
7F4A DD7E02        00660           LD      A,(IX+2)        ;GET SECTOR #
7F4D 32EE37        00670           LD      (37EEH),A       ;OUTPUT
7F50 C5            00680           PUSH    BC              ;WASTE TIME
7F51 C1            00690           POP     BC
7F52 21EC37        00700           LD      HL,37ECH        ;DISK ADDRESS
7F55 DD5E03        00710           LD      E,(IX+3)        ;PUT BUFFER ADDRESS IN DE
```

```
7F58 DD5604     00720          LD      D,(IX+4)
7F5B 3EBC       00730          LD      A,8CH          ;READ COMMAND
7F5D 77         00740          LD      (HL),A         ;OUTPUT
7F5E C5         00750          PUSH    BC             ;WASTE TIME
7F5F C1         00760          POP     BC
7F60 C5         00770          PUSH    BC
7F61 C1         00780          POP     BC
7F62 01EF37     00790          LD      BC,37EFH       ;DATA REG ADDRESS
7F65 7E         00800 REA040   LD      A,(HL)           ;GET STATUS
7F66 0F         00810          RRCA                     ;ALIGN
7F67 3008       00820          JR      NC,REA050        ;GO IF DONE
7F69 0F         00830          RRCA                     ;ALIGN
7F6A 30F9       00840          JR      NC,REA040        ;GO IF NOT DRQ
7F6C 0A         00850          LD      A,(BC)           ;GET BYTE
7F6D 12         00860          LD      (DE),A           ;STORE IN MEMORY
7F6E 13         00870          INC     DE               ;INCREMENT MEMORY PNTR
7F6F 18F4       00880          JR      REA040           ;LOOP TIL DONE
7F71 3AEC37     00890 REA050   LD      A,(37ECH)      ;GET STATUS
7F74 E61C       00900          AND     1CH            ;CHECK FOR PROPER STATUS
7F76 DD7706     00910 REA090   LD      (IX+6),A       ;STORE STATUS
7F79 DDE1       00920          POP     IX             ;RESTORE REGISTERS
7F7B E1         00930          POP     HL
7F7C D1         00940          POP     DE
7F7D C1         00950          POP     BC
7F7E F1         00960          POP     AF
7F7F C9         00970          RET                    ;RETURN TO CALLING PROG
0000            00980          END
00000 TOTAL ERRORS
```

READDS DECIMAL VALUES

```
245, 197, 213, 229, 221, 229, 205, 127, 10, 229,
221, 225, 221, 126, 0, 60, 71, 62, 128, 7,
16, 253, 50, 224, 55, 221, 126, 5, 183, 32,
8, 33, 0, 0, 43, 125, 180, 32, 251, 58,
236, 55, 203, 71, 32, 249, 221, 126, 1, 50,
239, 55, 197, 193, 62, 23, 50, 236, 55, 197,
193, 197, 193, 58, 236, 55, 203, 71, 32, 249,
230, 152, 32, 44, 221, 126, 2, 50, 238, 55,
197, 193, 33, 236, 55, 221, 94, 3, 221, 86,
4, 62, 140, 119, 197, 193, 197, 193, 1, 239,
55, 126, 15, 48, 8, 15, 48, 249, 10, 18,
19, 24, 244, 58, 236, 55, 230, 28, 221, 119,
6, 221, 225, 225, 209, 193, 241, 201
```

CHKSUM= 12

RESTDS: RESTORE DISK

System Configuration

Model I.

Description

RESTDS performs a restore operation on disk drive 1 through 4. The disk drive head is moved over track 0. RESTDS is an "initialization" procedure for READDS and WRDSEC to reset the disk to a known configuration.

Input/Output Parameters

On input, the L register contains the drive number of the disk drive to be used, 0 through 3 (corresponding to drives 1 through 4). The H register is set to 0 if a "wait after select" is to be done, or to a 1 if "no wait" is to occur. The wait is used if no current disk operation is taking place and the disk drive motor is not spinning.

On output, the disk head is restored over track 0. If the operation is successful, HL is returned with a zero result. If a disk error has occurred, HL is returned with a nonzero result.

INPUT

H	L
Ø=WAIT, 1=NO WAIT	DRIVE #,Ø-3

⟹

OUTPUT

H L
Ø=OK, ≠ Ø=ERROR

Algorithm

The disk drive number in L is first converted to the proper select configuration at RES010. The select byte is then output to disk memory-mapped address 37E0H to select one of the disk drives.

The wait bit is then examined. If this bit is a zero, the loop at RES015 counts HL through 65,536 counts to wait until the disk drive motor is up to speed before continuing.

The disk status is then examined (RES020). If the disk is not busy, a restore command (3) is sent to the disk controller command register at address 37ECH. A series of time-wasting instructions is then done.

The code at RES030 gets the disk status after completion of the restore, ANDs it with a "proper result" mask, and returns the status in HL.

Sample Calling Sequence

```
NAME OF SUBROUTINE? RESTDS
HL VALUE? Ø    WAIT, DRIVE Ø
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 38000
SUBROUTINE EXECUTED AT  38000
INPUT:              OUTPUT:
HL= Ø               HL= Ø   STATUS = OK


NAME OF SUBROUTINE?
```

Program Listing

```
7F00            00100           ORG     7F00H           ;0522
                00110 ;**********************************************************
                00120 ;* RESTORE DISK. PERFORMS A RESTORE OPERATION ON DISK.    *
                00130 ;*     INPUT: H=0 IF WAIT AFTER SELECT, 1 IF NO WAIT      *
                00140 ;*            L=DRIVE NUMBER, 0 - 3                       *
                00150 ;*     OUTPUT:HL=0 FOR OK, <>0 FOR ERROR                  *
                00160 ;**********************************************************
                00170 ;
7F00 F5         00180 RESTDS    PUSH    AF              ;SAVE REGISTERS
7F01 C5         00190           PUSH    BC
7F02 CD7F0A     00200           CALL    0A7FH           ;***GET DRIVE #***
7F05 7D         00210           LD      A,L             ;PUT IN A
7F06 3C         00220           INC     A               ;INCREMENT BY ONE
```

```
7F07 47         00230           LD      B,A             ;NOW IN B
7F08 3E80       00240           LD      A,80H           ;MASK FOR CONVERSION
7F0A 07         00250 RES010    RLCA                      ;CONVERT TO ADDRESS
7F0B 10FD       00260           DJNZ    RES010            ;LOOP 'TIL DONE
7F0D 32E037     00270           LD      (37E0H),A       ;SELECT DRIVE
7F10 7C         00280           LD      A,H             ;GET WAIT/NO WAIT
7F11 B7         00290           OR      A               ;TEST
7F12 2008       00300           JR      NZ,RES020       ;GO IF NO WAIT
7F14 210000     00310           LD      HL,0            ;WAIT COUNT
7F17 2B         00320 RES015    DEC     HL                ;DELAY LOOP 6
7F18 7D         00330           LD      A,L               ;TEST DONE 4
7F19 B4         00340           OR      H                 ;4
7F1A 20FB       00350           JR      NZ,RES015         ;LOOP UNTIL HL=0 7/12
7F1C 3AEC37     00360 RES020    LD      A,(37ECH)         ;GET STATUS
7F1F CB47       00370           BIT     0,A               ;TEST BUSY
7F21 20F9       00380           JR      NZ,RES020         ;GO IF BUSY
7F23 3E03       00390           LD      A,3             ;RESTORE COMMAND
7F25 32EC37     00400           LD      (37ECH),A       ;OUTPUT TO DISK
7F28 C5         00410           PUSH    BC              ;WASTE TIME
7F29 C1         00420           POP     BC
7F2A C5         00430           PUSH    BC
7F2B C1         00440           POP     BC
7F2C 3AEC37     00450 RES030    LD      A,(37ECH)         ;GET STATUS
7F2F CB47       00460           BIT     0,A               ;TEST BUSY
7F31 20F9       00470           JR      NZ,RES030         ;GO IF BUSY
7F33 E698       00480           AND     98H             ;TEST STATUS
7F35 6F         00490           LD      L,A             ;NOW IN A
7F36 2600       00500           LD      H,0             ;NOW IN HL
7F38 C1         00510           POP     BC              ;RESTORE REGISTERS
7F39 F1         00520           POP     AF
7F3A C39A0A     00530           JP      0A9AH           ;***RETURN STATUS***
7F3D C9         00540           RET                     ;NON-BASIC RETURN
0000            00550           END
00000 TOTAL ERRORS
```

RESTDS DECIMAL VALUES

```
245, 197, 205, 127, 10, 125, 60, 71, 62, 128,
7, 16, 253, 50, 224, 55, 124, 183, 32, 8,
33, 0, 0, 43, 125, 180, 32, 251, 58, 236,
55, 203, 71, 32, 249, 62, 3, 50, 236, 55,
197, 193, 197, 193, 58, 236, 55, 203, 71, 32,
249, 230, 152, 111, 38, 0, 193, 241, 195, 154,
10, 201
```

CHKSUM= 197

RKNOWT: READ KEYBOARD WITH NO WAIT

System Configuration

Model I, Model III.

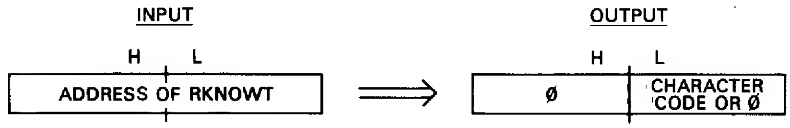
Description

RKNOWT reads the keyboard and returns immediately after scanning all keys to determine if a keypress has occurred. If a keypress has occurred, the subroutine returns with the key code; if no keypress has occurred, the subroutine returns with 0. The key position is converted to a code from a user-specified table of codes. Normally, these codes would be the ASCII codes for the keys on the keyboard, but the user may substitute his own codes for special key functions. Both upper- and lower-case keys are translated, and all keys are read including BREAK, CLEAR, up arrow, down arrow, right arrow, and left arrow.

Input/Output Parameters

On input, the HL register pair contains the address of RKNOWT. This address is the same as the USR location in BASIC or the address in the assembly-language call. It is used to make all of the code in RKNOWT relocatable.

On output, HL contains the keycode if a key was pressed, or 0 if no key was detected.



Algorithm

The basic problem in RKNOWT is to detect if a key is being pressed, and if it is, to convert its row–column coordinates into an index to a table to obtain the key code.

The table is at RKNTAB. RKNTAB is a 120-byte table that contains all the translation codes for the keys. The row arrangement is determined by the electrical connections to the keys, shown below. The first 56 bytes of the table represent keys with no SHIFT. There is a "gap" of 8 unused bytes to simplify coding, and then 56 additional bytes that represent keys with a SHIFT.

Keyboard layout and codes.

	BIT Ø	1	2	3	4	5	6	7		RKNOWT/RKWAIT HEXADECIMAL TABLE VALUES FOR STANDARD ASCII
ROW Ø	@	A	B	C	D	E	F	G	NO SHIFT	40,41,42,43,44,45,46,47
1	H	I	J	K	L	M	N	O		48,49,4A,4B,4C,4D,4E,4F
2	P	Q	R	S	T	U	V	W		50,51,52,53,54,55,56,57
3	X	Y	Z							58,59,5A,Ø,Ø,Ø,Ø,Ø
4	Ø	! 1	" 2	# 3	$ 4	% 5	& 6	' 7		30,31,32,33,34,35,36,37
5	(8	) 9	* :	+ ;	< ,	= -	> .	? /		38,39,3A,3B,2C,2D,2E,2F
6	ENTER	CLEAR	BREAK	↑	↓	←	→	SPACE		0D,2F,01,5B,5C,5D,5E,20
7	SHIFT								(GAP)	Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø

SHIFT

20,61,62,63,64,65,66,67
68,69,6A,6B,6C,6D,6E,6F
70,71,72,73,74,75,76,77
78,79,7A,Ø,Ø,Ø,Ø,Ø
20,21,22,23,24,25,26,27
28,29,2A,2B,3C,3D,3E,3F
0D,2F,01,5B,5C,5D,5E,20

The loop at RKN030 scans the seven rows of the keyboard and looks for a keypress in a row. The address of row 0 is 3801H, and this is initially put into HL. If no key is found in row 0, the L portion of the address is shifted left to produce an address in HL of 3802H. This process is repeated for the additional rows until all seven rows have been scanned, as evidenced by a one bit in bit 7 of L. If no key has been found (A register is a zero), a return with HL equal to zero is made at RKN090.

If any row is nonzero when read, RKN040 is entered. At this point, the row address of 3801H, 3802H, 3804H, etc., is in HL; the code at RKN050 converts this row address to a row number 0 to 7 times 8. This "index" of 0, 8, 16, 24, 32, 40, or 48 is saved.

The A register contains the column bits for the row. One column bit (or more for multiple key presses) is a one. The code at RKN070 converts the column bit into a column number of 7 to 0. This column number is then added to ROW*8.

Next, the SHIFT key is read by "LD A,(3880H)." The shift key bit is aligned and merged with COL+ ROW*8 to produce an index of SHIFT*64+ ROW*8+ COL. This index is then added to the start of RKNOWT and the displacement of the code table, RKNTAB, to point to a location within the table corresponding to the key pressed. The code just prior to RKN090 accesses the code table to pick up the proper code for the key that has been pressed. If multiple keys in the same row have been pressed, the rightmost key is detected and the others ignored.

Sample Calling Sequence

```
NAME OF SUBROUTINE? RKNOWT
HL VALUE? 36788   ADDRESS OF RKNOWT
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 36788
SUBROUTINE EXECUTED AT  36788
INPUT:              OUTPUT:
HL= 36788           HL= 0    NO KEY PRESSED

NAME OF SUBROUTINE?
```

Notes

1. The eight bytes between lower and upper case may contain any values.
2. The calling program must "time out" keyboard debounce.

Program Listing

```
7F00           ORG     7F00H          ;0522
         ;*******************************************************
         ;* READ KEYBOARD NO WAIT. READS KEYBOARD AND RETURNS   *
         ;* WITH NO WAIT.                                       *
         ;*     INPUT: HL=> ADDRESS OF RKWAIT                   *
         ;*     OUTPUT:HL=CHARACTER READ OR 0 IF NO KEY PRESSED *
         ;*******************************************************
         ;
```

```
7F00 F5         00180 RKNOWT  PUSH    AF              ;SAVE REGISTERS
7F01 C5         00190         PUSH    BC
7F02 DDE5       00200         PUSH    IX
7F04 CD7F0A     00210         CALL    0A7FH           ;***GET BASE ADDRESS***
7F07 E5         00220         PUSH    HL              ;TRANSFER TO IX
7F08 DDE1       00230         POP     IX
7F0A 210138     00240 RKN020  LD      HL,3801H          ;ADDRESS OF FIRST ROW
7F0D 7E         00250 RKN030  LD      A,(HL)              ;GET NEXT ROW
7F0E B7         00260         OR      A                   ;TEST FOR KEY
7F0F 200B       00270         JR      NZ,RKN040           ;GO IF KEY PRESS
7F11 CB25       00280         SLA     L                   ;GET NEXT ROW ADDRESS
7F13 CB7D       00290         BIT     7,L                 ;TEST FOR LAST ADDR
7F15 28F6       00300         JR      Z,RKN030            ;GO IF NOT LAST
7F17 210000     00310         LD      HL,0            ;0 FOR NO KEY
7F1A 1828       00320         JR      RKN090          ;GO TO RETURN
7F1C 4F         00330 RKN040  LD      C,A             ;SAVE COLUMN BITS
7F1D AF         00340         XOR     A               ;CLEAR COUNT
7F1E CB3D       00350 RKN050  SRL     L                 ;SHIFT OUT ROW ADDRESS
7F20 3804       00360         JR      C,RKN060          ;GO IF ONE BIT FOUND
7F22 C608       00370         ADD     A,8               ;ROW*8
7F24 18F8       00380         JR      RKN050            ;LOOP TIL DONE
7F26 06FF       00390 RKN060  LD      B,0FFH          ;INITIALIZE COUNT
7F28 04         00400 RKN070  INC     B                 ;FIND COLUMN BIT
7F29 CB39       00410         SRL     C                 ;SHIFT OUT COLUMNS
7F2B 30FB       00420         JR      NC,RKN070         ;LOOP 'TIL FOUND
7F2D 80         00430         ADD     A,B             ;ROW*8+COL
7F2E 4F         00440         LD      C,A             ;NOW IN C
7F2F 3A8038     00450         LD      A,(3880H)       ;GET SHIFT BIT
7F32 0F         00460         RRCA                    ;NOW IN BIT 7
7F33 0F         00470         RRCA                    ;NOW IN BIT 6
7F34 81         00480         ADD     A,C             ;SHIFT*64+ROW*8+COL
7F35 4F         00490         LD      C,A             ;INDEX TO C
7F36 0600       00500         LD      B,0             ;NOW IN BC
7F38 DD09       00510         ADD     IX,BC           ;BASE PLUS INDEX
7F3A 014C00     00520         LD      BC,RKNTAB       ;TRANSLATION TABLE
7F3D DD09       00530         ADD     IX,BC           ;BASE+INDEX+DISPL
7F3F DD6E00     00540         LD      L,(IX+0)        ;GET CHARACTER
7F42 2600       00550         LD      H,0             ;NOW IN HL
7F44 DDE1       00560 RKN090  POP     IX              ;RESTORE REGISTERS
7F46 C1         00570         POP     BC
7F47 F1         00580         POP     AF
7F48 C39A0A     00590         JP      0A9AH           ;***RETURN WITH ARGUMENT***
7F4B C9         00600         RET                     ;NON-BASIC RETURN
004C            00610 RKNTAB  EQU     $-RKNOWT        ;TRANSLATION TABLE
0008            00620         DEFS    8               ;NO SHIFT ROW 0
0008            00630         DEFS    8               ;             1
0008            00640         DEFS    8               ;             2
0008            00650         DEFS    8               ;             3
0008            00660         DEFS    8               ;             4
0008            00670         DEFS    8               ;             5
0008            00680         DEFS    8               ;             6
0008            00690         DEFS    8               ;NOT USED
0008            00700         DEFS    8               ;SHIFT ROW    0
0008            00710         DEFS    8               ;             1
0008            00720         DEFS    8               ;             2
0008            00730         DEFS    8               ;             3
0008            00740         DEFS    8               ;             4
0008            00750         DEFS    8               ;             5
0008            00760         DEFS    8               ;             6
0000            00770         END
00000 TOTAL ERRORS
```

RKNOWT DECIMAL VALUES

245, 197, 221, 229, 205, 127, 10, 229, 221, 225,
33, 1, 56, 126, 183, 32, 11, 203, 37, 203,
125, 40, 246, 33, 0, 0, 24, 40, 79, 175,

```
203, 61, 56, 4, 198, 8, 24, 248, 6, 255,
4, 203, 57, 48, 251, 128, 79, 58, 128, 56,
15, 15, 129, 79, 6, 0, 221, 9, 1, 76,
0, 221, 9, 221, 110, 0, 38, 0, 221, 225,
193, 241, 195, 154, 10, 201
```

```
CHKSUM= 29
```

RKWAIT: READ KEYBOARD AND WAIT

System Configuration

Model I, Model III.

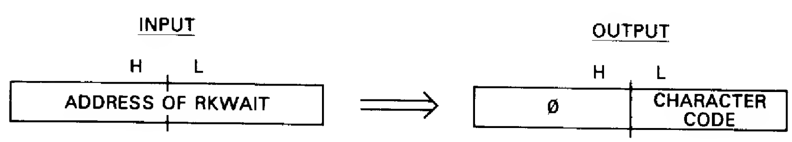
Description

RKWAIT reads the keyboard and returns after a key has been pressed. The key position is converted to a code from a user-specified table of codes. Normally, these codes would be the ASCII codes for the keys on the keyboard, but the user may substitute his own codes for special key functions. Both upper- and lower-case keys are translated, and all keys are read including BREAK, CLEAR, up arrow, down arrow, right arrow, and left arrow.

Input/Output Parameters

On input, the HL register pair contains the address of RKWAIT. This address is the same as the USR location in BASIC or the address in the assembly-language call. It is used to make all the code in RKWAIT relocatable.

On output, HL contains the keycode.



Algorithm

The basic problem in RKWAIT is to detect if a key is being pressed and if it is, to convert its row column coordinates into an index to a table to obtain the key code.

The table is at RKWTAB. RKWTAB is a 120-byte table that contains all the translation codes for the keys. The row arrangement is determined by the electrical connections to the keys, shown below. The first 56 bytes of the table represent keys with no SHIFT. There is a "gap" of 8 unused bytes to simplify coding, and then 56 additional bytes that represent keys with a SHIFT.

BIT

	Ø	1	2	3	4	5	6	7		RKNOWT/RKWAIT HEXADECIMAL TABLE VALUES FOR STANDARD ASCII
ROW Ø	@	A	B	C	D	E	F	G	NO SHIFT	40,41,42,43,44,45,46,47
1	H	I	J	K	L	M	N	O		48,49,4A,4B,4C,4D,4E,4F
2	P	Q	R	S	T	U	V	W		50,51,52,53,54,55,56,57
3	X	Y	Z							58,59,5A,Ø,Ø,Ø,Ø,Ø
4	Ø	! 1	" 2	# 3	$ 4	% 5	& 6	' 7		30,31,32,33,34,35,36,37
5	(8	) 9	* :	+ ;	< ,	= -	> .	? /		38,39,3A,3B,2C,2D,2E,2F
6	ENTER	CLEAR	BREAK	↑	↓	←	→	SPACE		0D,2F,01,5B,5C,5D,5E,20
7	SHIFT								(GAP)	Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø

Keyboard layout and codes.

SHIFT

20,61,62,63,64,65,66,67
68,69,6A,6B,6C,6D,6E,6F
70,71,72,73,74,75,76,77
78,79,7A,Ø,Ø,Ø,Ø,Ø
20,21,22,23,24,25,26,27
28,29,2A,2B,3C,3D,3E,3F
0D,2F,01,5B,5C,5D,5E,20

The loop at RKW030 scans the seven rows of the keyboard and looks for a keypress in a row. The address of row 0 is 3801H, and this is initially put into HL. If no key is found in row 0, the L portion of the address is shifted left to produce an address in HL of 3802H. This process is repeated for the additional rows until all seven rows have been scanned, as evidenced by a one bit in bit 7 of L. If no key has been found after seven rows, a loop is made back to RKW020 to repeat the scan.

If any row is nonzero when read, RKN040 is entered. At this point, the row address of 3801H, 3802H, 3804H, etc., is in HL; the code at RKW050 converts this row address to a row number of 0 to 7 times 8. This "index" of 0, 8, 16, 24, 32, 40, or 48 is saved.

The A register contains the column bits for the row. One (or more for multiple key presses) is a one. The code at RKN070 converts the column bit into a column number of 7 to 0. This column number is then added to ROW*8.

Next, the SHIFT key is read by "LD A,(3880H)." The shift key bit is aligned and merged with COL+ ROW*8 to produce an index of SHIFT*64+ ROW*8+ COL.

At this point a "debounce delay" of 50 milliseconds is performed. This ensures that the key is not reread if RKWAIT is reentered immediately by the calling program.

The index is then added to the start of RKWAIT and the displacement of the code table, RKWTAB, to point to a location within the table corresponding to the key pressed. The code just prior to RKW090 accesses the code table to pick up the proper code for the key that has been pressed. If multiple keys in the same row have been pressed, the rightmost key is detected and the others ignored.

Sample Calling Sequence

```
NAME OF SUBROUTINE? RKWAIT
HL VALUE? 38000    ADDRESS OF RKWAIT
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 38000
SUBROUTINE EXECUTED AT  38000
INPUT:              OUTPUT:
HL= 38000           HL= 65   "A" KEY, NO SHIFT

NAME OF SUBROUTINE?
```

Notes

1. The eight bytes between lower and upper case may contain any values.
2. The debounce delay may be adjusted as required. A 50 millisecond delay is about 20 characters per second or 240 words per minute. Change locations 7F33H and 7F34H to alter the debounce delay.

Program Listing

```
7F00           00100           ORG     7F00H           ;0522
               00110 ;**************************************************
               00120 ;* READ KEYBOARD AND WAIT. READS KEYBOARD AND WAITS   *
               00130 ;* UNTIL KEY PRESS.                                   *
               00140 ;*     INPUT: HL=> ADDRESS OF RKWAIT                  *
               00150 ;*     OUTPUT:HL=CHARACTER READ                       *
               00160 ;**************************************************
               00170 ;
7F00 F5        00180 RKWAIT    PUSH    AF              ;SAVE REGISTERS
7F01 C5        00190           PUSH    BC
7F02 DDE5      00200           PUSH    IX
7F04 CD7F0A    00210           CALL    0A7FH           ;***GET BASE ADDRESS***
7F07 E5        00220           PUSH    HL              ;TRANSFER TO IX
7F08 DDE1      00230           POP     IX
7F0A 210138    00240 RKW020    LD      HL,3801H        ;ADDRESS OF FIRST ROW
7F0D 7E        00250 RKW030    LD      A,(HL)          ;GET NEXT ROW
7F0E B7        00260           OR      A               ;TEST FOR KEY
7F0F 2008      00270           JR      NZ,RKW040       ;GO IF KEY PRESS
7F11 CB25      00280           SLA     L               ;GET NEXT ROW ADDRESS
7F13 CB7D      00290           BIT     7,L             ;TEST FOR LAST ADDR
7F15 28F6      00300           JR      Z,RKW030        ;GO IF NOT LAST
7F17 18F1      00310           JR      RKW020          ;LAST-LOOP 'TIL KEY
7F19 4F        00320 RKW040    LD      C,A             ;SAVE COLUMN BITS
7F1A AF        00330           XOR     A               ;CLEAR COUNT
7F1B CB3D      00340 RKW050    SRL     L               ;SHIFT OUT ROW ADDRESS
7F1D 3804      00350           JR      C,RKW060        ;GO IF ONE BIT FOUND
7F1F C608      00360           ADD     A,8             ;ROW*8
7F21 18F8      00370           JR      RKW050          ;LOOP TIL DONE
7F23 06FF      00380 RKW060    LD      B,0FFH          ;INITIALIZE COUNT
7F25 04        00390 RKW070    INC     B               ;FIND COLUMN BIT
7F26 CB39      00400           SRL     C               ;SHIFT OUT COLUMNS
7F28 30FB      00410           JR      NC,RKW070       ;LOOP 'TIL FOUND
7F2A 80        00420           ADD     A,B             ;ROW*8+COL
7F2B 4F        00430           LD      C,A             ;NOW IN C
7F2C 3A8038    00440           LD      A,(3880H)       ;GET SHIFT BIT
7F2F 0F        00450           RRCA                    ;NOW IN BIT 7
7F30 0F        00460           RRCA                    ;NOW IN BIT 6
7F31 81        00470           ADD     A,C             ;SHIFT*64+ROW*8+COL
7F32 21100F    00480           LD      HL,3856         ;DELAY COUNT (50 MS)
7F35 01FFFF    00490           LD      BC,-1           ;DECREMENT VALUE
```

```
7F38 09         00500 RKW080  ADD    HL,BC          ;DELAY FOR BOUNCE 11
7F39 38FD       00510         JR     C,RKW080       ;LOOP 'TIL HL NEG 7/12
7F3B 4F         00520         LD     C,A           ;INDEX TO C
7F3C 0600       00530         LD     B,0           ;NOW IN BC
7F3E DD09       00540         ADD    IX,BC         ;BASE PLUS INDEX
7F40 015200     00550         LD     BC,RKWTAB     ;TRANSLATION TABLE
7F43 DD09       00560         ADD    IX,BC         ;BASE+INDEX+DISPL
7F45 DD6E00     00570         LD     L,(IX+0)      ;GET CHARACTER
7F48 2600       00580         LD     H,0           ;NOW IN HL
7F4A DDE1       00590         POP    IX            ;RESTORE REGISTERS
7F4C C1         00600         POP    BC
7F4D F1         00610         POP    AF
7F4E C39A0A     00620         JP     0A9AH         ;***RETURN WITH ARGUMENT***
7F51 C9         00630         RET                  ;NON-BASIC RETURN
0052            00640 RKWTAB  EQU    $-RKWAIT      ;TRANSLATION TABLE
0008            00650         DEFS   8             ;NO SHIFT ROW 0
0008            00660         DEFS   8             ;             1
0008            00670         DEFS   8             ;             2
0008            00680         DEFS   8             ;             3
0008            00690         DEFS   8             ;             4
0008            00700         DEFS   8             ;             5
0008            00710         DEFS   8             ;             6
0008            00720         DEFS   8             ;NOT USED
0008            00730         DEFS   8             ;SHIFT ROW    0
0008            00740         DEFS   8             ;             1
0008            00750         DEFS   8             ;             3
0008            00760         DEFS   8             ;             3
0008            00770         DEFS   8             ;             4
0008            00780         DEFS   8             ;             5
0008            00790         DEFS   8             ;             6
0000            00800         END
00000 TOTAL ERRORS
```

```
RKWAIT DECIMAL VALUES

 245, 197, 221, 229, 205, 127, 10, 229, 221, 225,
 33, 1, 56, 126, 183, 32, 8, 203, 37, 203,
 125, 40, 246, 24, 241, 79, 175, 203, 61, 56,
 4, 198, 8, 24, 248, 6, 255, 4, 203, 57,
 48, 251, 128, 79, 58, 128, 56, 15, 15, 129,
 33, 16, 15, 1, 255, 255, 9, 56, 253, 79,
 6, 0, 221, 9, 1, 82, 0, 221, 9, 221,
 110, 0, 38, 0, 221, 225, 193, 241, 195, 154,
 10, 201

CHKSUM= 69
```

SCDOWN: SCROLL SCREEN DOWN

System Configuration

Model I, Model III.

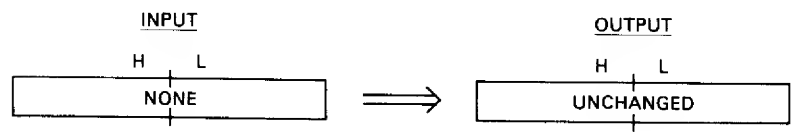
Description

SCDOWN scrolls the video display down one line. Scrolling down causes lines 1 through 15 to be moved up into line positions 0 through 14. Scrolling can be used in displaying text or data that cannot be displayed in the 1024 bytes of one video screen.

When scrolling down, line 15 is blanked in preparation for displaying the next line "below" the screen.

Input/Output Parameters

There are no input or output parameters. A call to SCDOWN simply causes a scroll down of one line, with a return to the calling program immediately following.



Algorithm

Scrolling is easily and efficiently handled by use of the Z-80 "block move" instructions. The LDIR moves a block of data from one area of memory to another, transferring the data "beginning to end" (lower-valued memory locations to higher-valued memory locations) of each block, one byte at a time.

The LDIR automatically transfers video memory bytes to locations 64 bytes "down" in memory. A total of 960 bytes are transferred as the first line "disappears."

After the transfer, the last line has been moved up to the second to last line, but still remains on the bottom of the screen. This line is "blanked" by a fill of 64 bytes of blank characters at SCD010.

Sample Calling Sequence

```
NAME OF SUBROUTINE? SCDOWN
HL VALUE?
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 36666
SUBROUTINE EXECUTED AT  36666
INPUT:              OUTPUT:


NAME OF SUBROUTINE?
```

Program Listing

```
7F00                      ORG     7F00H           ;0522
                   ;**********************************************************
                   ;* SCROLL SCREEN DOWN. SCROLLS SCREEN DOWN ONE LINE.      *
                   ;*     INPUT: NONE                                         *
                   ;*     OUTPUT:SCREEN SCROLLED DOWN                         *
                   ;**********************************************************
                   ;
7F00 F5            SCDOWN PUSH    AF              ;SAVE REGISTERS
7F01 C5                   PUSH    BC
7F02 D5                   PUSH    DE
7F03 E5                   PUSH    HL
7F04 21403C               LD      HL,3C40H        ;SOURCE
```

```
7F07 11003C    00220          LD      DE,3C00H        ;DESTINATION
7F0A 01C003    00230          LD      BC,960          ;# OF BYTES
7F0D EDB0      00240          LDIR                      ;SCROLL
7F0F 21C03F    00250          LD      HL,3FC0H        ;LINE TO BE BLANKED
7F12 3E20      00260          LD      A,' '           ;LOAD BLANK CHARACTER
7F14 0640      00270          LD      B,64            ;64 CHARACTERS ON LINE
7F16 77        00280 SCD010   LD      (HL),A            ;STORE BLANK IN LINE
7F17 23        00290          INC     HL                ;BUMP POINTER
7F18 10FC      00300          DJNZ    SCD010            ;LOOP IF NOT DONE
7F1A E1        00310          POP     HL              ;RESTORE REGISTERS
7F1B D1        00320          POP     DE
7F1C C1        00330          POP     BC
7F1D F1        00340          POP     AF
7F1E C9        00350          RET                     ;RETURN
0000           00360          END
00000 TOTAL ERRORS
```

```
SCDOWN DECIMAL VALUES


 245, 197, 213, 229, 33, 64, 60, 17, 0, 60,
 1, 192, 3, 237, 176, 33, 192, 63, 62, 32,
 6, 64, 119, 35, 16, 252, 225, 209, 193, 241,
 201

CHKSUM= 86
```

SCUSCR: SCROLL SCREEN UP

System Configuration

Model I, Model III.

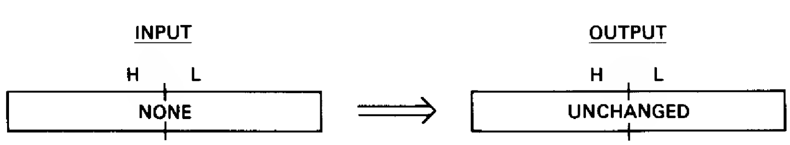
Description

SCUSCR scrolls the video display up one line. Scrolling up causes lines 0 through 14 to be moved down into line positions 1 through 15. Scrolling can be used in displaying text or data that cannot be displayed in the 1024 bytes of one video screen.

When scrolling up, line 0 is blanked in preparation for displaying the next line "above" the screen.

Input/Output Parameters

There are no input or output parameters. A call to SCUSCR simply causes a scroll up of one line, with a return to the calling program immediately following.



Algorithm

Scrolling is easily and efficiently handled by use of the Z-80 "block move" instructions. The LDDR moves a block of data from one area of memory to another, transferring the data "end to beginning" (higher-valued memory locations to lower-valued memory locations) of each block, one byte at a time.

The LDDR automatically transfers video memory bytes to locations 64 bytes "up" in memory. A total of 960 bytes are transferred as the last line "disappears."

After the transfer, the first line has been moved down to the second line, but still remains on the top of the screen. This line is "blanked" by a fill of 64 bytes of blank characters at SCU010.

Sample Calling Sequence

```
NAME OF SUBROUTINE? SCUSCR
HL VALUE?
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 41111
SUBROUTINE EXECUTED AT  41111
INPUT:           OUTPUT:

NAME OF SUBROUTINE?
```

Program Listing

```
7F00            00100          ORG     7F00H           ;0522
                00110 ;****************************************************
                00120 ;* SCROLL SCREEN UP. SCROLLS SCREEN UP ONE LINE.    *
                00130 ;*     INPUT: NONE                                  *
                00140 ;*     OUTPUT:SCREEN SCROLLED UP                    *
                00150 ;****************************************************
                00160 ;
7F00 F5         00170 SCUSCR   PUSH    AF              ;SAVE REGISTERS
7F01 C5         00180          PUSH    BC
7F02 D5         00190          PUSH    DE
7F03 E5         00200          PUSH    HL
7F04 21803F     00210          LD      HL,3F80H        ;SOURCE
7F07 11C03F     00220          LD      DE,3FC0H        ;DESTINATION
7F0A 01C003     00230          LD      BC,960          ;# OF BYTES
7F0D EDB8       00240          LDDR                      ;SCROLL
7F0F 21003C     00250          LD      HL,3C00H        ;LINE TO BE BLANKED
7F12 3E20       00260          LD      A,' '           ;LOAD BLANK CHARACTER
7F14 0640       00270          LD      B,64            ;64 CHARACTERS ON LINE
7F16 77         00280 SCU010   LD      (HL),A            ;STORE BLANK IN LINE
7F17 23         00290          INC     HL                ;BUMP POINTER
7F18 10FC       00300          DJNZ    SCU010            ;LOOP IF NOT DONE
7F1A E1         00310          POP     HL              ;RESTORE REGISTERS
7F1B D1         00320          POP     DE
7F1C C1         00330          POP     BC
7F1D F1         00340          POP     AF
7F1E C9         00350          RET                     ;RETURN
0000            00360          END
00000 TOTAL ERRORS
```

SCUSCR DECIMAL VALUES

```
245, 197, 213, 229, 33, 128, 63, 17, 192, 63,
1, 192, 3, 237, 184, 33, 0, 60, 62, 32,
6, 64, 119, 35, 16, 252, 225, 209, 193, 241,
201
```

CHKSUM= 161

SDASCI: SCREEN DUMP TO PRINTER IN ASCII

Configuration

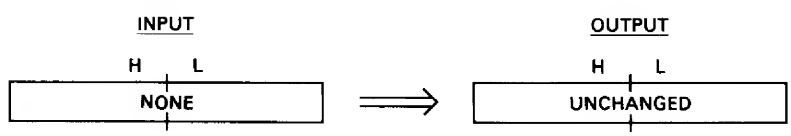
Model I, Model III.

Description

SDASCI dumps the contents of the video display to the system line printer. SDASCI may be called at any time to record the contents of the screen. ASCII characters are printed as they appear on the screen. Graphics characters are printed as a period. The system line printer must be able to print 64 character positions across. The screen is printed as 16 lines of 64 characters.

Input/Output Parameters

There are no input parameters. The screen contents are printed and a return to the calling program is done.



Algorithm

The HL register pair holds the current screen location starting from 3C00H, the screen start. The B register is used to hold the number of characters per line, 64. It is decremented down to zero so that a carriage return at the end of line can be made to the system line printer.

There are two loops. The main loop starts at SDA005. The inner loop handles each screen line and starts at SDA010. For each new line, the line character count of 64 is placed into the B register at SDA005.

In the SDA010 loop, a character is loaded into A from the next character position. Bit 7 of the character is tested. If this bit is a one, a period is substituted for the graphics character. If the character is not a graphics character (SDA020), a 20H is subtracted from the character and bit 7 is tested. If bit 7 is set, the value of the character is less than 20H, and 40H is added to compensate for the lower case option. The character is then saved in the stack while a status check is made of the line printer.

The code at SDA050 checks line printer status. When the line printer is ready, the character is popped from the stack and printed. The HL pointer is then incremented by one, and the line character count in B decremented. If B is zero, a carriage return is output to the line printer for the end of the line by a jump back to SDA040.

SDA060 tests for a condition of -1 in the B register. If this is true, a carriage return has just been output, and a test is made for HL=4000H, which marks the end of the dump. If H is not equal to 40H, a jump is made back to SDA005 to output the next line. If there is not a -1 in B at SDA060, the current line is still being processed and a jump is made back to SDA010 for the next character in the line.

Sample Calling Sequence

```
NAME OF SUBROUTINE? SDASCI
HL VALUE?
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 40000
TRS-80 ASSEMBLY LANGUAGE SUBROUTINES EXERCISER
```

```
NAME OF SUBROUTINE? SDASCI
HL VALUE?
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO
? 40000
```

16 SCREEN LINES

```
SUBROUTINE EXECUTED AT   40000
INPUT:              OUTPUT:

NAME OF SUBROUTINE?
```

Notes

1. If this subroutine is used for the Model III, make the following change in the listing: Substitute "OUT (0F8H),A" for "LD (37E8H),A". Replace the corresponding decimal values of "50, 232, 55" with decimal values of "211, 248, 0".

Program Listing

```
7F00              ORG     7F00H           ;0520
      ;*****************************************************
      ;* SCREEN DUMP TO PRINTER. CAUSES CONTENTS OF SCREEN TO *
      ;* BE DUMPED TO THE SYSTEM LINE PRINTER. GRAPHICS ARE   *
      ;* PRINTED AS A PERIOD.                                 *
      ;*     INPUT: NONE                                      *
      ;*     OUTPUT:SCREEN CONTENTS PRINTED                   *
      ;*****************************************************
      ;
```

```
7F00 F5         00190 SDASCI  PUSH   AF           ;SAVE REGISTERS
7F01 C5         00200         PUSH   BC
7F02 E5         00210         PUSH   HL
7F03 21003C     00220         LD     HL,3C00H     ;SCREEN START ADDRESS
7F06 0640       00230 SDA005  LD     B,64         ;# OF CHARACTERS/LINE
7F08 7E         00240 SDA010  LD     A,(HL)       ;GET NEXT SCREEN BYTE
7F09 CB7F       00250         BIT    7,A          ;TEST FOR GRAPHICS
7F0B 2804       00260         JR     Z,SDA020     ;GO IF GRAPHICS BYTE
7F0D 3E2E       00270         LD     A,'.'        ;PERIOD FOR GRAPHICS
7F0F 180A       00280         JR     SDA040       ;GO TO PRINT
7F11 D620       00290 SDA020  SUB    20H          ;TEST FOR CONTROL
7F13 CB7F       00300         BIT    7,A          ;CONTROL IF SET
7F15 2802       00310         JR     Z,SDA030     ;GO IF NOT LT 20H
7F17 C640       00320         ADD    A,40H        ;ADJUST FOR CONTROL
7F19 C620       00330 SDA030  ADD    A,20H        ;RESTORE FOR SUB
7F1B F5         00340 SDA040  PUSH   AF           ;SAVE CHARACTER
7F1C 3AE837     00350 SDA050  LD     A,(37E8H)    ;GET PRINTER STATUS
7F1F E6F0       00360         AND    0F0H         ;MASK OUT UNUSED BITS
7F21 FE30       00370         CP     30H          ;TEST STATUS
7F23 20F7       00380         JR     NZ,SDA050    ;GO IF BUSY
7F25 F1         00390         POP    AF           ;RESTORE CHARACTER
7F26 32E837     00400         LD     (37E8H),A    ;PRINT CHARACTER
7F29 23         00410         INC    HL           ;BUMP SCREEN POINTER
7F2A 05         00420         DEC    B            ;DECREMENT CHAR CNT
7F2B 78         00430         LD     A,B          ;GET COUNT
7F2C B7         00440         OR     A            ;TEST
7F2D 2004       00450         JR     NZ,SDA060    ;GO IF NOT 0
7F2F 3E0D       00460         LD     A,13         ;END OF LINE
7F31 18E8       00470         JR     SDA040       ;OUTPUT CR
7F33 FEFF       00480 SDA060  CP     0FFH         ;TEST FOR -1
7F35 20D1       00490         JR     NZ,SDA010    ;STILL IN LINE
7F37 2B         00500         DEC    HL           ;ADJUST FOR FALSE INC
7F38 7C         00510         LD     A,H          ;JUST PRINTED CR
7F39 FE40       00520         CP     40H          ;AT END OF SCREEN?
7F3B 20C9       00530         JR     NZ,SDA005    ;GO IF NO
7F3D E1         00540         POP    HL           ;RESTORE REGISTERS
7F3E C1         00550         POP    BC
7F3F F1         00560         POP    AF
7F40 C9         00570         RET                 ;RETURN TO CALLING PROG
0000            00580         END
00000 TOTAL ERRORS
```

```
SDASCI DECIMAL VALUES

245, 197, 229, 33, 0, 60, 6, 64, 126, 203,
127, 40, 4, 62, 46, 24, 10, 214, 32, 203,
127, 40, 2, 198, 64, 198, 32, 245, 58, 232,
55, 230, 240, 254, 48, 32, 247, 241, 50, 232,
55, 35, 5, 120, 183, 32, 4, 62, 13, 24,
232, 254, 255, 32, 209, 43, 124, 254, 64, 32,
201, 225, 193, 241, 201

CHKSUM= 163
```

SDGRAP: SCREEN DUMP TO PRINTER IN GRAPHICS

Configuration

Model I, Model III.

Description

SDGRAP dumps the contents of the video display to the system line printer. SDGRAP may be called at any time to record the contents of the screen. Graphics characters are printed as they appear on the screen by an "O." ASCII characters are not printed. The system line printer must be able to print 128 character positions across. The screen is printed as 48 rows of 128 pixels.

Input/Output Parameters

There are no input parameters. The screen contents are printed and a return to the calling program is done.

INPUT		OUTPUT
H L		H L
NONE	⟹	UNCHANGED

Algorithm

The SDGRAP subroutine uses an internal print subroutine at SDG050. This subroutine first tests the current character position contents in the A register for graphics. If the current contents are nongraphics (ASCII), a blank character is used for the print; if the current contents are graphics, an "O" is used for the print. The blank or "O" is then saved in the stack.

Next in the print subroutine, a test is made for printer status. The code at SDG060 loops until the printer is not busy. When the printer is ready, the blank or "O" character is output. The print subroutine then adjusts a "bit mask" in the B register. This mask represents the current bit position in the character position being tested. Each graphics character has six bit positions, bits 5 through 0. The bit mask is shifted left one bit to mask the next bit position. Finally, the print subroutine tests for the return point. There are three return points. If bits 0, 2, or 4 have just been printed, a return is made to SDG030. If bits 1, 3, or 5 have just been printed, a return is made to SDG035. If neither of these conditions is present (B equals zero), a carriage return has just been printed and a return is made to SDG040. The normal subroutine structure is not used so that all code in SDGRAP can be relocatable.

The main code in SDGRAP uses three loops. The outermost loop (SDG010) handles character positions, in sets of three graphics rows. The next innermost loop handles the three rows within each character position. The innermost loop handles each row of graphics bits.

Each set of three rows (one line) starts off with the mask bit in B set for pixel 0. The character is picked up via the pointer in HL. SDG050 is called to output the first pixel. The B mask is now set to pixel 1. SDG050 is again called for pixel 1. Next, (SDG035), the line pointer in HL, is bumped, and the bit mask is shifted back to the right two bit positions. For the first row, B would now hold 1. Now a test is made of HL. If HL is not at the end of line, the next character is picked up and pixels 0 and 1 printed. If HL is at the end of line, a carriage return is printed by a call to SDG050, and the bit mask in B is shifted left two bit positions. If the first row had just been printed, B would now contain a 4. HL is now adjusted to point back to the beginning of the line by adding −64. If the next row is still within a character position, a loop back to SDG012 prints the next row.

If the next row starts a new line, the pointer in HL is bumped by 64 to point to the next line of three rows. A test is made for HL=4000H, signifying that all rows have been printed. If this is not the case, a jump is made back to SDG010 to print the next set of three rows.

Sample Calling Sequence

```
NAME OF SUBROUTINE? SDGRAP
HL VALUE?
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 38888
```

48 SCREEN ROWS

```
SUBROUTINE EXECUTED AT  38888
INPUT:          OUTPUT:


NAME OF SUBROUTINE?
```

Notes

1. ASCII characters on the screen are ignored, but will not cause erroneous results.

2. The dimensions of the printout on many printers will be 12.8 inches horizontal by 8 inches vertical, which will be approximately the "aspect ratio" of the screen.

3. If this subroutine is used for the Model III, make the following change in the listing: Substitute "OUT (0F8H),A" for "LD (37E8H),A." Replace the corresponding decimal values of "50, 232, 55" with decimal values of "211, 248, 0."

Program Listing

```
7F00                   ORG     7F00H          ;0520
               ;**********************************************************
               ;* GRAPHICS DUMP TO PRINTER. CAUSES CONTENTS OF SCREEN    *
               ;* TO BE DUMPED TO SYSTEM LINE PRINTER AS 128 BY 48 MAT-*
               ;* RIX OF OS. TEXT IS IGNORED.                            *
               ;*      INPUT: NONE                                       *
               ;*      OUTPUT:SCREEN CONTENTS PRINTED                    *
               ;**********************************************************
               ;
7F00 F5        SDGRAP  PUSH    AF             ;SAVE REGISTERS
7F01 C5                PUSH    BC
7F02 D5                PUSH    DE
7F03 E5                PUSH    HL
7F04 21003C            LD      HL,3C00H       ;START OF SCREEN
7F07 0601      SDG010  LD      B,1              ;MASK BIT FOR UPPER LEFT
7F09 C5        SDG012  PUSH    BC               ;SAVE MASK
7F0A C1        SDG015  POP     BC               ;GET MASK
7F0B 7E        SDG020  LD      A,(HL)               ;GET CHARACTER
7F0C 182E              JR      SDG050               ;OUTPUT LFT SIDE BIT
7F0E 7E        SDG030  LD      A,(HL)                ;GET CHARACTER
7F0F 182B              JR      SDG050                ;OUTPUT RIGHT SIDE BIT
7F11 23        SDG035  INC     HL                    ;BUMP LINE POINTER
7F12 CB38              SRL     B                     ;ADJUST BACK MASK
7F14 CB38              SRL     B
7F16 C5                PUSH    BC                    ;SAVE MASK
7F17 7D                LD      A,L                   ;GET CHAR POS ADDR
7F18 E63F              AND     3FH                   ;TEST FOR 64TH CHAR
7F1A 20EE              JR      NZ,SDG015             ;GO IF NOT END OF LINE
7F1C 47                LD      B,A                 ;0 TO B
7F1D 3E0D              LD      A,13                ;CARRIAGE RETURN
7F1F 1826              JR      SDG054              ;PRINT
7F21 C1        SDG040  POP     BC                  ;RESTORE BIT MASK
7F22 CB20              SLA     B                   ;NEXT LINE MASK
7F24 CB20              SLA     B
7F26 11C0FF            LD      DE,-64              ;FOR RTN TO LINE START
7F29 19                ADD     HL,DE               ;RESET TO LINE START
7F2A CB70              BIT     6,B                 ;TEST FOR THREE LINES
7F2C 28DB              JR      Z,SDG012            ;GO IF NOT THREE
7F2E 114000            LD      DE,64             ;FOR NEXT SCREEN LINE
7F31 19                ADD     HL,DE             ;POINT TO NEXT SCREEN LINE
7F32 7C                LD      A,H               ;GET MS BYTE OF ADDRESS
7F33 FE40              CP      40H               ;TEST FOR END OF SCREEN
7F35 20D0              JR      NZ,SDG010         ;GO IF NOT END
7F37 E1                POP     HL             ;RESTORE REGISTERS
7F38 D1                POP     DE
7F39 C1                POP     BC
7F3A F1                POP     AF
7F3B C9                RET                    ;RETURN TO CALLING PROGRAM
               ; PRINT SUBROUTINE
```

```
7F3C CB7F       00590 SDG050  BIT     7,A          ;TEST FOR NON-GRAPHICS
7F3E 2801       00600         JR      Z,SDG052     ;GO IF NON-GRAPHICS
7F40 A0         00610         AND     B            ;GET GRAPHICS BIT
7F41 3E20       00620 SDG052  LD      A,' '        ;BLANK
7F43 2802       00630         JR      Z,SDG054     ;GO IF BIT RESET
7F45 3E4F       00640         LD      A,'O'        ;BIT SET
7F47 F5         00650 SDG054  PUSH    AF           ;SAVE CHARACTER
7F48 3AE837     00660 SDG060  LD      A,(37E8H)    ;GET PRINTER STATUS
7F4B E6F0       00670         AND     0F0H         ;MASK OUT INACTIVE BITS
7F4D FE30       00680         CP      30H          ;TEST FOR STATUS
7F4F 20F7       00690         JR      NZ,SDG060    ;LOOP IF BUSY
7F51 F1         00700         POP     AF           ;RESTORE CHARACTER
7F52 32E837     00710         LD      (37E8H),A    ;OUTPUT CHARACTER
7F55 CB20       00720         SLA     B            ;ADJUST BIT MASK
7F57 78         00730         LD      A,B          ;GET BIT MASK
7F58 E6AA       00740         AND     0AAH         ;TEST FOR RETURN
7F5A 20B2       00750         JR      NZ,SDG030    ;RETURN FOR RIGHT SIDE
7F5C 78         00760         LD      A,B          ;GET BIT MASK
7F5D E654       00770         AND     54H          ;TEST FOR RETURN
7F5F 20B0       00780         JR      NZ,SDG035    ;RETURN FOR NEXT ROW
7F61 18BE       00790         JR      SDG040       ;RETURN FOR LINE
0000            00800         END
00000 TOTAL ERRORS
```

```
SDGRAP DECIMAL VALUES

245, 197, 213, 229, 33, 0, 60, 6, 1, 197,
193, 126, 24, 46, 126, 24, 43, 35, 203, 56,
203, 56, 197, 125, 230, 63, 32, 238, 71, 62,
13, 24, 38, 193, 203, 32, 203, 32, 17, 192,
255, 25, 203, 112, 40, 219, 17, 64, 0, 25,
124, 254, 64, 32, 208, 225, 209, 193, 241, 201,
203, 127, 40, 1, 160, 62, 32, 40, 2, 62,
79, 245, 58, 232, 55, 230, 240, 254, 48, 32,
247, 241, 50, 232, 55, 203, 32, 120, 230, 170,
32, 178, 120, 230, 84, 32, 176, 24, 190

CHKSUM= 64
```

SETCOM: SET RS-232-C INTERFACE

System Configuration

Model I.

Description

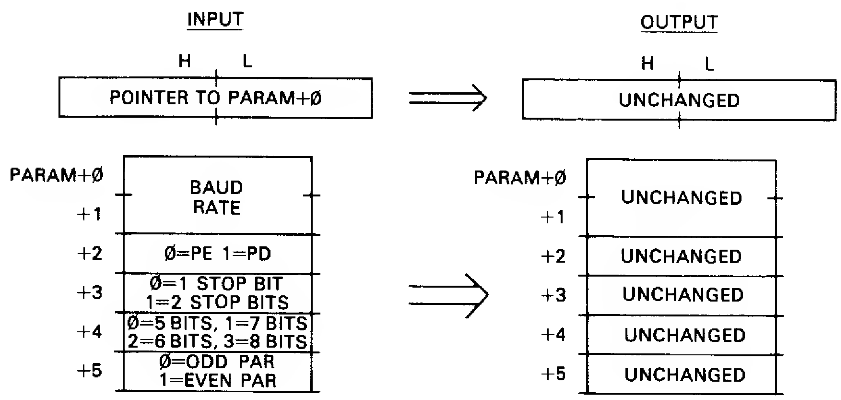
SETCOM programs the RS-232-C controller in lieu of setting the switches on the RS-232-C controller board. (SETCOM must be run before the NECDRV program can be used.)

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block are the baud rate for which the RS-232-C interface is to be set, 110, 150, 300, 600, 1200, 2400, 4800, or 9600. The next byte is set to a zero if parity is to be enabled, or to a one if parity is to be disabled.

The next byte of the parameter block is set to a zero if one stop bit is to be used, or to a one if two stop bits are to be used. The next byte contains the number of bits in the RS-232-C transfer; 0 is 5 bits, 1 is 7 bits, 2 is 6 bits, or 3 is 8 bits. The next byte contains a zero if odd parity is to be used, or a one if even parity is to be used.

On output, the parameter block remains unchanged, and the RS-232-C interface is initialized.



Algorithm

The SETCOM subroutine reads the parameters, merges, and aligns them into the proper format for the RS-232-C controller, and writes them out to the controller.

First, the controller is reset by an "OUT (0E8H),A." Next, the parity type is picked up into A and shifted to yield 00000P00. Next, the number of bits is merged, and shifted to yield 0000PNN0. Next, the number of stop bits is merged and shifted to yield 000PNNS0. Next, the parity enable/disable bit is merged and shifted to yield PNNSP000. Next, the BRK and RTS bits are set and the PNNSP101 configuration is output to port address 0EAH.

The next portion of code converts the baud rate to the proper RS-232-C code. To keep the code relocatable, "linear" code (not table lookup) is used. The least significant byte of the baud rate is picked up and compared to the ls byte of 110, 150, 300, etc. The proper code is then output to port address 0E9H.

Sample Calling Sequence

```
NAME OF SUBROUTINE? SETCOM
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0  2  1200   1200 BAUD
+ 2  1  1      PD
+ 3  1  0      ONE STOP BIT
+ 4  1  1      SEVEN BITS
+ 5  1  0      ODD PARITY
+ 6  0  0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 39000
```

```
SUBROUTINE EXECUTED AT  39000
INPUT:              OUTPUT:
HL= 40000           HL= 40000
PARAM+ 0  176       PARAM+ 0  176
PARAM+ 1  4         PARAM+ 1  4
PARAM+ 2  1         PARAM+ 2  1   UNCHANGED
PARAM+ 3  0         PARAM+ 3  0
PARAM+ 4  1         PARAM+ 4  1
PARAM+ 5  0         PARAM+ 5  0

NAME OF SUBROUTINE?
```

Notes

1. No check is made on proper parameters in the parameter block.

2. The OR prior to 0EAH output may be modified as required to set a different configuration of BRK, DTR, RTS.

3. Note transposed order of number of bits.

Program Listing

```
7F00            00100         ORG     7F00H           ;0522
                00110 ;*****************************************************
                00120 ;* SET RS-232-C. PROGRAMS THE RS-232-C CONTROLLER.   *
                00130 ;*    INPUT: HL=> PARAMETER BLOCK                     *
                00140 ;*           PARAM+0,+1=BAUD RATE - 110, 150, 300, 600, *
                00150 ;*                  1200, 2400, 4800, 9600           *
                00160 ;*           PARAM+2=0=PARITY ENABLED, 1=PARITY DISAB *
                00170 ;*           PARAM+3=0=ONE STOP BIT, 1=TWO STOP BITS  *
                00180 ;*           PARAM+4=0=5 BITS, 1=7 BITS, 2=6 BITS, 3=8 *
                00190 ;*                  BITS                              *
                00200 ;*           PARAM+5=0=ODD PARITY, 1=EVEN             *
                00210 ;*    OUTPUT:RS-232-C CONTROLLER INITIALIZED          *
                00220 ;*****************************************************
                00230 ;
7F00 F5         00240 SETCOM  PUSH    AF              ;SAVE REGISTERS
7F01 E5         00250         PUSH    HL
7F02 DDE5       00260         PUSH    IX
7F04 CD7F0A     00270         CALL    0A7FH           ;***GET PB LOC'N***
7F07 E5         00280         PUSH    HL              ;TRANSFER TO IX
7F08 DDE1       00290         POP     IX
7F0A D3E8       00300         OUT     (0E8H),A        ;RESET RS-232-C
7F0C DD7E05     00310         LD      A,(IX+5)        ;PARITY
7F0F 07         00320         RLCA                    ;ALIGN
7F10 07         00330         RLCA
7F11 DDB604     00340         OR      (IX+4)          ;MERGE # BITS
7F14 07         00350         RLCA                    ;ALIGN
7F15 DDB603     00360         OR      (IX+3)          ;# OF STOP BITS
7F18 07         00370         RLCA                    ;ALIGN
7F19 DDB602     00380         OR      (IX+2)          ;PARITY ENAB/DIS
7F1C 07         00390         RLCA                    ;ALIGN
7F1D 07         00400         RLCA
7F1E 07         00410         RLCA
7F1F F605       00420         OR      5               ;SET BRK, RTS
7F21 D3EA       00430         OUT     (0EAH),A        ;OUTPUT
7F23 DD7E00     00440         LD      A,(IX+0)        ;GET LSB OF BAUD RATE
7F26 FE6E       00450         CP      110             ;110?
7F28 2004       00460         JR      NZ,SET010       ;GO IF NO
7F2A 3E22       00470         LD      A,22H           ;110 CODE
7F2C 1832       00480         JR      SET080          ;GO TO SET
7F2E FE96       00490 SET010  CP      150             ;150?
7F30 2004       00500         JR      NZ,SET020       ;GO IF NO
```

```
7F32 3E44      00510         LD      A,44H           ;150 CODE
7F34 182A      00520         JR      SET080          ;GO TO SET
7F36 FE2C      00530 SET020  CP      44              ;300?
7F38 2004      00540         JR      NZ,SET030       ;GO IF NO
7F3A 3E55      00550         LD      A,55H           ;300 CODE
7F3C 1822      00560         JR      SET080          ;GO TO SET
7F3E FE58      00570 SET030  CP      88              ;600?
7F40 2004      00580         JR      NZ,SET040       ;GO IF NO
7F42 3E66      00590         LD      A,66H           ;600 CODE
7F44 181A      00600         JR      SET080          ;GO TO SET
7F46 FEB0      00610 SET040  CP      176             ;1200?
7F48 2004      00620         JR      NZ,SET050       ;GO IF NO
7F4A 3E77      00630         LD      A,77H           ;1200 CODE
7F4C 1812      00640         JR      SET080          ;GO TO SET
7F4E FE60      00650 SET050  CP      96              ;2400?
7F50 2004      00660         JR      NZ,SET060       ;GO IF NO
7F52 3EAA      00670         LD      A,0AAH          ;2400 CODE
7F54 180A      00680         JR      SET080          ;GO TO SET
7F56 FEC0      00690 SET060  CP      192             ;4800?
7F58 2004      00700         JR      NZ,SET070       ;GO IF NO
7F5A 3ECC      00710         LD      A,0CCH          ;4800 CODE
7F5C 1802      00720         JR      SET080          ;GO TO SET
7F5E 3EEE      00730 SET070  LD      A,0EEH          ;9600 CODE
7F60 32E900    00740 SET080  LD      (0E9H),A        ;OUTPUT TO BRG
7F63 DDE1      00750         POP     IX              ;RESTORE REGISTERS
7F65 E1        00760         POP     HL
7F66 F1        00770         POP     AF              ;RETURN TO CALLING PROG
7F67 C9        00780         RET
0000           00790         END
00000 TOTAL ERRORS
```

SETCOM DECIMAL VALUES

```
245, 229, 221, 229, 205, 127, 10, 229, 221, 225,
211, 232, 221, 126, 5, 7, 7, 221, 182, 4,
7, 221, 182, 3, 7, 221, 182, 2, 7, 7,
7, 246, 5, 211, 234, 221, 126, 0, 254, 110,
32, 4, 62, 34, 24, 50, 254, 150, 32, 4,
62, 68, 24, 42, 254, 44, 32, 4, 62, 85,
24, 34, 254, 88, 32, 4, 62, 102, 24, 26,
254, 176, 32, 4, 62, 119, 24, 18, 254, 96,
32, 4, 62, 170, 24, 10, 254, 192, 32, 4,
62, 204, 24, 2, 62, 238, 50, 233, 0, 221,
225, 225, 241, 201
```

CHKSUM= 186

SOIARR: SEARCH ONE-DIMENSIONAL INTEGER ARRAY

System Configuration

Model I, Model III, Model II Stand Alone.

Description

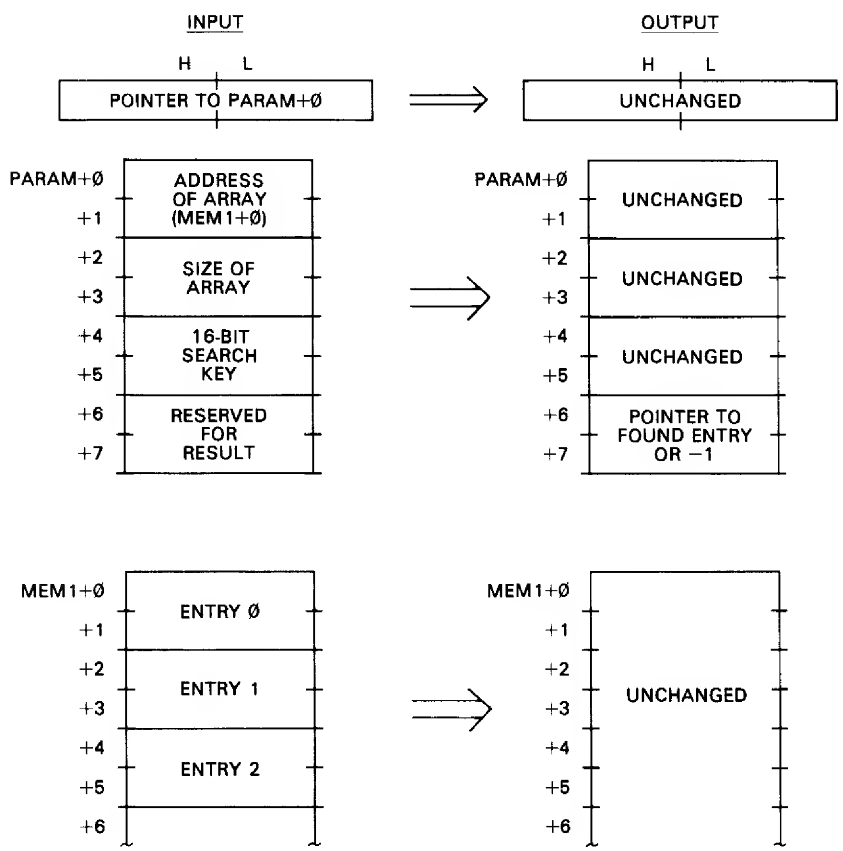
SOIARR searches a BASIC or other one-dimensional integer array for a given 16-bit search key. The array may be any size within memory limits. The array is assumed to be made up of 16-bit entries. SOIARR returns the address of the entry matching the search key, or a −1 if no entry matches the search key.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the 16-bit address of the array, arranged in standard Z-80 address format, least significant byte followed by most significant byte. The next two bytes of the array contain the number of entries in the array. (Note that this value is one-half the number of bytes in the array!)

The next two bytes contain the 16-bit search key. The arrangement of the search key may correspond to the arrangement of data in the array. If the array is a BASIC array, the data in the search key will be least significant byte followed by most significant byte; if the array is made up of two ASCII characters arranged first and second, then the search key should have the same arrangement. The last two bytes are reserved for the result of the search.

On output, PARAM+6, +7 holds the address of the entry corresponding to the search key, or −1 if no entry has been found.



Algorithm

The SOIARR scans the array one entry (two bytes) at a time from beginning to end, looking for the search key. The number of entries is put into BC, the starting address of the array into IY, and the search key in DE. HL is used as a working register for the compare of the entries to the key.

The loop at SOI010 performs the scan. The next entry is put into HL. The search key in DE is then subtracted from HL. If the result is zero, the current address in IY is returned in HL. If the result is nonzero, no match occurred, and the code at SOI020 increments IY by two to point to the next entry, and then decrements the count of entries in BC. A test is then made of BC; if it is zero, all entries have been tested and a "not found" return is made. If there are additional entries to be tested, a loop back to SOI010 is done.

Sample Calling Sequence

```
NAME OF SUBROUTINE? SOIARR
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0  2  45000   ADDRESS OF ARRAY
+ 2  2  5       5 ENTRIES (10 BYTES)
+ 4  2  1234    SEARCH KEY
+ 6  2  0
+ 8  0  0
MEMORY BLOCK 1 LOCATION? 45000
MEMORY BLOCK 1 VALUES?
+ 0  2  2345 ┐
+ 2  2  3456 │
+ 4  2  5678 ├5 ENTRY ARRAY (TABLE)
+ 6  2  6789 │
+ 8  2  1234 ┘
+ 10  0  0
MEMORY BLOCK 2 LOCATION?
MOVE SUBROUTINE TO? 38000
SUBROUTINE EXECUTED AT  38000
INPUT:                OUTPUT:
HL= 40000             HL= 40000
PARAM+ 0  200         PARAM+ 0  200┐
PARAM+ 1  175         PARAM+ 1  175│
PARAM+ 2  5           PARAM+ 2  5  ├UNCHANGED
PARAM+ 3  0           PARAM+ 3  0  │
PARAM+ 4  210         PARAM+ 4  210│
PARAM+ 5  4           PARAM+ 5  4  ┘
PARAM+ 6  0           PARAM+ 6  208┐FOUND AT 45008
PARAM+ 7  0           PARAM+ 7  175┘
MEMB1+ 0  41          MEMB1+ 0  41 ┐
MEMB1+ 1  9           MEMB1+ 1  9  │
MEMB1+ 2  128         MEMB1+ 2  128│
MEMB1+ 3  13          MEMB1+ 3  13 │
MEMB1+ 4  46          MEMB1+ 4  46 │
MEMB1+ 5  22          MEMB1+ 5  22 ├UNCHANGED
MEMB1+ 6  133         MEMB1+ 6  133│
MEMB1+ 7  26          MEMB1+ 7  26 │
MEMB1+ 8  210         MEMB1+ 8  210│
MEMB1+ 9  4           MEMB1+ 9  4  ┘

NAME OF SUBROUTINE?
```

Notes

1. "Array" in this case corresponds to a table of two-byte entries.

```
7F00               00100         ORG     7F00H           ;0522
                   00110 ;****************************************************
                   00120 ;* SEARCH ONE-D INTEGER ARRAY. SEARCHES INTEGER ARRAY   *
                   00130 ;* FOR SPECIFIED SEARCH KEY.                            *
                   00140 ;*     INPUT: HL=> PARAMETER BLOCK                      *
                   00150 ;*            PARAM+0,+1=ADDRESS OF ARRAY               *
                   00160 ;*            PARAM+2,+3=SIZE OF ARRAY                  *
                   00170 ;*            PARAM+4,+5=16-BIT SEARCH KEY              *
                   00180 ;*            PARAM+6,+7=RESERVED FOR RESULT OF SEARCH  *
                   00190 ;*     OUTPUT:PARAM+6,+7 HOLDS ADDRESS IF KEY FOUND OR  *
                   00200 ;*            -1 OTHERWISE                              *
                   00210 ;****************************************************
                   00220 ;
7F00 F5            00230 SOIARR  PUSH    AF              ;SAVE REGISTERS
7F01 C5            00240         PUSH    BC
7F02 D5            00250         PUSH    DE
7F03 E5            00260         PUSH    HL
7F04 DDE5          00270         PUSH    IX
7F06 FDE5          00280         PUSH    IY
7F08 CD7F0A        00290         CALL    0A7FH           ;***GET PB LOC'N***
7F0B E5            00300         PUSH    HL              ;TRANSFER TO IX
7F0C DDE1          00310         POP     IX
7F0E DD4E02        00320         LD      C,(IX+2)        ;PUT SIZE IN BC
7F11 DD4603        00330         LD      B,(IX+3)
7F14 DD6E00        00340         LD      L,(IX+0)        ;PUT ADDRESS IN HL
7F17 DD6601        00350         LD      H,(IX+1)
7F1A DD5E04        00360         LD      E,(IX+4)        ;PUT KEY IN DE
7F1D DD5605        00370         LD      D,(IX+5)
7F20 E5            00380         PUSH    HL              ;ARRAY ADDRESS TO IY
7F21 FDE1          00390         POP     IY
7F23 FD6E00        00400 SOI010  LD      L,(IY+0)          ;GET NEXT ARRAY ENTRY
7F26 FD6601        00410         LD      H,(IY+1)
7F29 B7            00420         OR      A                 ;CLEAR CARRY
7F2A ED52          00430         SBC     HL,DE             ;TEST FOR EQUALITY
7F2C 2005          00440         JR      NZ,SOI020         ;GO IF NOT FOUND
7F2E FDE5          00450         PUSH    IY                ;TRANSFER IY TO HL
7F30 E1            00460         POP     HL
7F31 180C          00470         JR      SOI030            ;GO TO RETURN
7F33 FD23          00480 SOI020  INC     IY                ;INCREMENT ARRAY LOC'N
7F35 FD23          00490         INC     IY
7F37 0B            00500         DEC     BC                ;DECREMENT COUNT
7F38 79            00510         LD      A,C               ;TEST COUNT
7F39 B0            00520         OR      B
7F3A 20E7          00530         JR      NZ,SOI010         ;LOOP IF COUNT NOT 0
7F3C 21FFFF        00540         LD      HL,-1           ;'NOT FOUND' FLAG
7F3F DD7506        00550 SOI030  LD      (IX+6),L        ;STORE LOC'N OR NOT FOUND
7F42 DD7407        00560         LD      (IX+7),H
7F45 FDE1          00570         POP     IY              ;RESTORE REGISTERS
7F47 DDE1          00580         POP     IX
7F49 E1            00590         POP     HL
7F4A D1            00600         POP     DE
7F4B C1            00610         POP     BC
7F4C F1            00620         POP     AF
7F4D C9            00630         RET                     ;RETURN TO CALLING PROG
0000               00640         END
00000 TOTAL ERRORS
```

SOIARR DECIMAL VALUES

245, 197, 213, 229, 221, 229, 253, 229, 205, 127,
10, 229, 221, 225, 221, 78, 2, 221, 70, 3,

```
221, 110, 0, 221, 102, 1, 221, 94, 4, 221,
86, 5, 229, 253, 225, 253, 110, 0, 253, 102,
1, 183, 237, 82, 32, 5, 253, 229, 225, 24,
12, 253, 35, 253, 35, 11, 121, 176, 32, 231,
33, 255, 255, 221, 117, 6, 221, 116, 7, 253,
225, 221, 225, 225, 209, 193, 241, 201

CHKSUM= 17
```

SPCAST: SERIAL PRINTER FROM CASSETTE

System Configuration

Model I, Model III.

Description

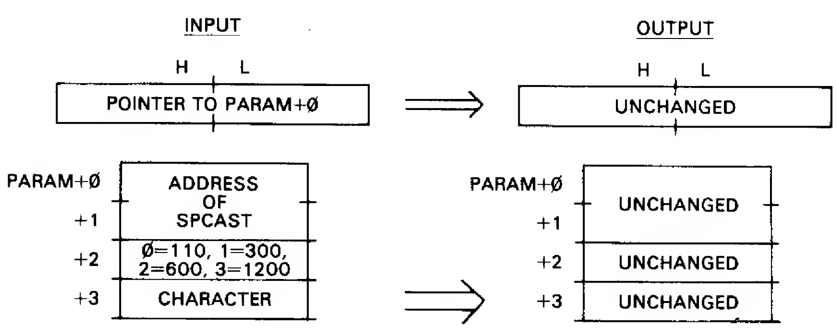
SPCAST uses the cassette output port to implement output to a serial printer. Additional external "hardware" is required to convert the cassette voltage levels to levels compatible with serial printers. A character at a time is output with a baud rate of 110, 300, 600, or 1200.

The format for output is one start bit, seven or eight data bits, and one stop bit with no parity. If the character to be output is a seven-bit ASCII character, the most significant bit should be set to zero, and the result will be seven data bits with two stop bits. If the character to be output is an eight-bit character, the result will be eight data bits with one stop bit.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the address of SPCAST, in standard Z-80 address format. The next byte contains a baud rate code of 0, 1, 2, or 3, corresponding to 110, 300, 600, or 1200 baud. The next byte contains the character to be output.

On output, the character has been transmitted. The parameter block remains unchanged.



Algorithm

SPCAST must take the given character and "strip off" the eight bits, translating each into a serial bit, which is sent out to the serial printer through the cassette port. The timing for each "bit time" is determined by the specified baud rate.

SPCAST first outputs a cassette off code by outputting a 2 to port 0FFH. Next, the baud rate code is obtained from the second byte of the parameter block. The code is multiplied by two and added to the start address of SPCAST and the table displacement. The result now points to a timing value in BAUDTB which represents the "bit time" for the given baud rate. This two-byte value is picked up and put into DE.

The cassette port is now turned on by outputting a 1 to 0FFH. This is the "start" bit. The count in DE is put into HL and the delay loop at SPC010 delays for one bit time.

The code at SPC015 is the main output loop of SPCAST. It loops eight times. For each loop, a bit from the character in C is shifted out into the carry. If the bit is a 0, a 2 level is output to port 0FFH; if the bit is a 1, a 1 level is output to port 0FFH. The second-level loop at SPC030 delays one bit time by decrementing the delay count in HL. If eight iterations have not been performed, another bit is transmitted.

The loop at SPC040 outputs a "stop" bit and delays for one bit time to terminate the transmission of the character.

Sample Calling Sequence

```
NAME OF SUBROUTINE? SPCAST
HL VALUE? 39000
PARAMETER BLOCK LOCATION? 39000
PARAMETER BLOCK VALUES?
+ 0   2   37000   ADDRESS OF SPCAST
+ 2   1   1       BAUD RATE = 300
+ 3   1   65      "A" TO BE OUTPUT
+ 4   0   0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 37000
SUBROUTINE EXECUTED AT  37000
INPUT:                OUTPUT:
HL= 39000             HL= 39000
PARAM+ 0   136        PARAM+ 0   136 ┐
PARAM+ 1   144        PARAM+ 1   144 ├ UNCHANGED
PARAM+ 2   1          PARAM+ 2   1   │
PARAM+ 3   65         PARAM+ 3   65  ┘

NAME OF SUBROUTINE?
```

Notes

1. External electronics must convert the cassette signal levels to RS-232-C compatible levels. The output signal level for a logic 0 is approximately 0 volts.

The output signal level for a logic 1 is approximately 0.85 volts. Corresponding RS-232-C signal levels are +3 volts or more for a logic 0 and −3 volts or less for a logic 1.

2. Multiply the BAUDTB values by 1.143 for a Model III.

Program Listing

```
7F00                   ORG     7F00H           ;0522
            ;******************************************************
            ;* SERIAL PRINTER FROM CASSETTE. OUTPUTS A CHARACTER TO *
            ;* A SERIAL PRINTER USING THE CPU CASSETTE PORT          *
            ;*    INPUT :HL=> PARAMETER BLOCK                        *
            ;*            PARAM+0,+1=ADDRESS OF SPCAST               *
            ;*            PARAM+2=BAUD RATE CODE 0=110, 1=300,       *
            ;*                 2=600, 3=1200                         *
            ;*            PARAM+3=CHARACTER TO BE OUTPUT             *
            ;*    OUTPUT:CHARACTER OUTPUT TO PRINTER                 *
            ;******************************************************
            ;
7F00 F5     SPCAST  PUSH    AF              ;SAVE REGISTERS
7F01 C5             PUSH    BC
7F02 D5             PUSH    DE
7F03 E5             PUSH    HL
7F04 DDE5           PUSH    IX
7F06 CD7F0A         CALL    0A7FH           ;***GET PB LOC'N***
7F09 E5             PUSH    HL              ;TRANSFER TO IX
7F0A DDE1           POP     IX
7F0C 3E01           LD      A,1             ;CASSETTE ON CODE
7F0E D3FF           OUT     (0FFH),A        ;SPACING
7F10 DD6E02         LD      L,(IX+2)        ;GET RATE CODE
7F13 2600           LD      H,0             ;NOW IN HL
7F15 29             ADD     HL,HL           ;CODE*2
7F16 DD5E00         LD      E,(IX+0)        ;ADDRESS OF THIS CODE
7F19 DD5601         LD      D,(IX+1)
7F1C 19             ADD     HL,DE           ;START+CODE
7F1D 115900         LD      DE,BAUDTB       ;TABLE DISPLACEMENT
7F20 19             ADD     HL,DE           ;POINT TO TIMING COUNT
7F21 5E             LD      E,(HL)          ;GET MS BYTE
7F22 23             INC     HL              ;POINT TO NEXT BYTE
7F23 56             LD      D,(HL)          ;GET LS BYTE
7F24 D5             PUSH    DE              ;COUNT TO HL
7F25 E1             POP     HL
7F26 3E02           LD      A,2             ;CASSETTE OFF CODE
7F28 D3FF           OUT     (0FFH),A        ;TURN OFF CASSETTE FOR SP
7F2A 2B     SPC010  DEC     HL                ;DECREMENT COUNT 6
7F2B 7C             LD      A,H               ;TEST COUNT 4
7F2C B5             OR      L                 ;TEST FOR ZERO 4
7F2D 20FB           JR      NZ,SPC010         ;GO IF NOT BIT TIME 7/12
7F2F DD4E03         LD      C,(IX+3)        ;GET CHARACTER
7F32 0608           LD      B,8             ;ITERATION COUNT
7F34 D5     SPC015  PUSH    DE                ;TRANSFER COUNT TO HL
7F35 E1             POP     HL
7F36 3E02           LD      A,2               ;CASSETTE OFF CODE
7F38 CB39           SRL     C                 ;SHIFT OUT BIT
7F3A 3002           JR      NC,SPC020         ;GO IF ZERO
7F3C 3E01           LD      A,1               ;CASSETTE ON CODE
7F3E D3FF   SPC020  OUT     (0FFH),A          ;OUTPUT TO CASSETTE
7F40 2B     SPC030  DEC     HL                  ;DECREMENT COUNT
7F41 7C             LD      A,H                 ;TEST COUNT
7F42 B5             OR      L
7F43 20FB           JR      NZ,SPC030           ;GO IF NOT DONE
7F45 10ED           DJNZ    SPC015            ;GO IF MORE BITS
7F47 D5             PUSH    DE              ;TRANSFER COUNT TO HL
7F48 E1             POP     HL
7F49 3E01           LD      A,1             ;CASSETTE ON CODE
```

```
7F4B D3FF      00680         OUT     (0FFH),A        ;OUTPUT TO CASSETTE
7F4D 2B        00690 SPC040  DEC     HL                ;DECREMENT COUNT
7F4E 7C        00700         LD      A,H               ;TEST COUNT
7F4F B5        00710         OR      L
7F50 20FB      00720         JR      NZ,SPC040         ;GO IF CNT NOT ZERO
7F52 DDE1      00730         POP     IX              ;RESTORE REGISTERS
7F54 E1        00740         POP     HL
7F55 D1        00750         POP     DE
7F56 C1        00760         POP     BC
7F57 F1        00770         POP     AF
7F58 C9        00780         RET                     ;RETURN
0059           00790 BAUDTB  EQU     $-SPCAST        ;BAUD COUNT TABLE
7F59 6C02      00800         DEFW    620             ;110
7F5B E300      00810         DEFW    227             ;300
7F5D 7200      00820         DEFW    114             ;600
7F5F 3900      00830         DEFW    57              ;1200
0000           00840         END
00000 TOTAL ERRORS
```

SPCAST DECIMAL VALUES

```
245, 197, 213, 229, 221, 229, 205, 127, 10, 229,
221, 225, 62, 1, 211, 255, 221, 110, 2, 38,
0, 41, 221, 94, 0, 221, 86, 1, 25, 17,
89, 0, 25, 94, 35, 86, 213, 225, 62, 2,
211, 255, 43, 124, 181, 32, 251, 221, 78, 3,
6, 8, 213, 225, 62, 2, 203, 57, 48, 2,
62, 1, 211, 255, 43, 124, 181, 32, 251, 16,
237, 213, 225, 62, 1, 211, 255, 43, 124, 181,
32, 251, 221, 225, 225, 209, 193, 241, 201, 108,
2, 227, 0, 114, 0, 57, 0
```

CHKSUM= 15

SQROOT: SQUARE ROOT

System Configuration

Model I, Model III, Model II Stand Alone.

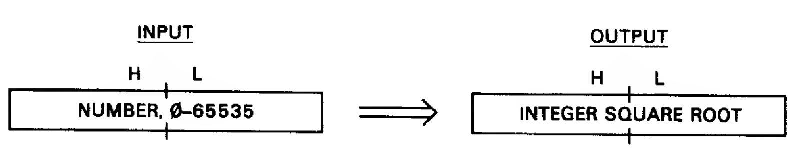
Description

SQROOT calculates the integer square root of a given 16-bit number. For example, if the number is 30,000, the subroutine will return 54 as the square root in place of 54.77.

Input/Output Parameters

On input, HL contains the "square," the number whose square root is to be found.

On output, HL contains the integer portion of the square root.



Algorithm

The SQROOT subroutine performs the square root operation by using the widely-known fact that the square root of any number is equal to the number of odd integers contained in the square. The square of 17, for example, contains 1 + 3+ 5 + 7 = 16. The total number of odd integers is 4, and this is the integer square root contained in 17.

The B register is initialized with a count of −1; B will count the number of odd integers in the square. DE is initialized with −1; DE will hold the negated value of the next odd integer—−1, −3, −5, and so forth.

The loop at SQR010 successively subtracts an odd integer from the original number by the "ADD HL,DE." The count of odd numbers in B is incremented with every subtract. The loop is terminated when the "residue" goes negative and the carry flag is reset after the add. At that point, the count of odd numbers is returned in HL.

Sample Calling Sequence

```
NAME OF SUBROUTINE? SQROOT
HL VALUE? 65535     SQUARE ROOT IS 255.99 . . .
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 55000
SUBROUTINE EXECUTED AT  55000
INPUT:              OUTPUT:
HL= 65535           HL= 255  INTEGER VALUE OF SQUARE ROOT

NAME OF SUBROUTINE?
```

Notes

1. The square may be "scaled-up" to achieve more precision. For example, if the square root of a number less than 100 is to be found, multiply the number by 256. The square root will then represent 16 times the actual square root. For example, 99 times 256 = 25344. The square root returned by the subroutine will be 159. This represents 159/16 or 9 and 15/16 or 9.9375, much closer to the actual square root of 9.949.

2. The square input in HL is an "unsigned" number. The maximum square can be 65,535.

Program Listing

```
7F00                          ORG     7F00H           ;0522
                ;*********************************************************
                ;* SQUARE ROOT. CALCULATES INTEGER PORTION OF SQUARE     *
                ;* ROOT OF A GIVEN NUMBER.                               *
                ;*     INPUT: HL=NUMBER                                  *
                ;*     OUTPUT:HL=INTEGER PORTION OF SQUARE RT OF NUMBER  *
                ;*********************************************************
                ;
7F00 C5         SQROOT        PUSH    BC              ;SAVE REGISTERS
7F01 D5                       PUSH    DE
7F02 CD7F0A                   CALL    0A7FH           ;***GET NUMBER***
```

```
7F05 06FF      00210          LD     B,0FFH         ;INITIALIZE RESULT
7F07 11FFFF    00220          LD     DE,-1          ;FIRST ODD SUBTRAHEND
7F0A 04        00230 SQR010   INC    B                ;INCREMENT RESULT COUNT
7F0B 19        00240          ADD    HL,DE            ;SUBTRACT ODD NUMBER
7F0C 1B        00250          DEC    DE               ;FIND NEXT ODD NUMBER
7F0D 1B        00260          DEC    DE
7F0E 38FA      00270          JR     C,SQR010         ;CONTINUE IF NOT MINUS
7F10 68        00280          LD     L,B            ;GET RESULT
7F11 2600      00290          LD     H,0            ;NOW IN HL
7F13 D1        00300          POP    DE             ;RESTORE REGISTERS
7F14 C1        00310          POP    BC
7F15 C39A0A    00320          JP     0A9AH          ;***RETURN ARGUMENT***
7F18 C9        00330          RET                   ;NON-BASIC RETURN
0000           00340          END
00000 TOTAL ERRORS
```

```
SQROOT DECIMAL VALUES


197, 213, 205, 127, 10, 6, 255, 17, 255, 255,
4, 25, 27, 27, 56, 250, 104, 38, 0, 209,
193, 195, 154, 10, 201


CHKSUM= 217
```

SROARR: SORT ONE-DIMENSIONAL INTEGER ARRAY

System Configuration

Model I, Model III, Model II Stand Alone.

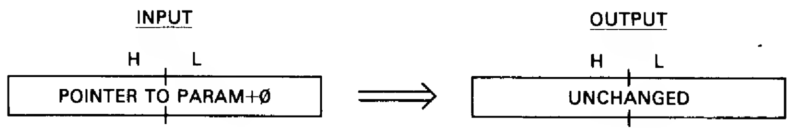
Description

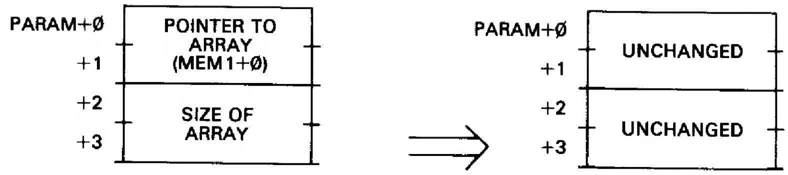
SROARR sorts a BASIC or other one-dimensional integer array. The array may be any size within memory limits. The array is assumed to be made up of 16-bit entries. SROARR arranges the entries in the array in ascending order based on their binary weight on a sixteen bit "unsigned" basis. In this scheme an entry of 8000H will be after an entry of 7FFFH. A "bubble sort" is used which requires no additional memory buffer other than the array itself.

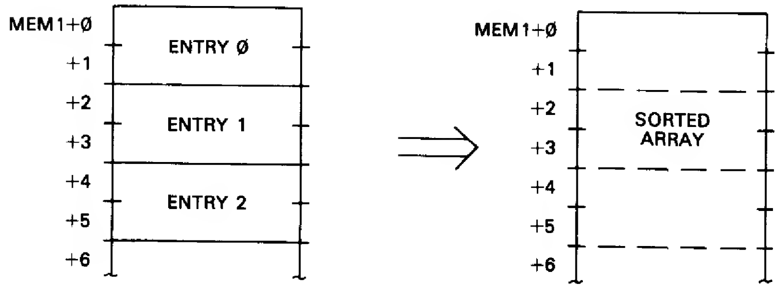
Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the 16-bit address of the array, arranged in standard Z-80 address format, least significant byte followed by most significant byte. The next two bytes of the array contain the number of entries in the array. (Note that this value is one-half the number of bytes in the array!)

On output, the array has been sorted in memory. The parameter block remains unchanged.







Algorithm

The SROARR sorts the entries by a bubble sort. This sort scans the array from bottom to top, moving one entry at a time. Each entry is compared to the next entry. If the top entry is a higher value than the next entry, the two entries are swapped, otherwise the entries are left unchanged. The next entry is then compared in the same fashion until all entries in the array have been examined. At the end of the scan, a "swap" flag is examined. If a swap occurred, another pass is made through the array. If no swap occurred, the array is sorted. A number of passes through the array may have to be made to sort the entries.

There are two loops in SROARR. The innermost loop controls the scan from top to bottom for every pass and starts at SRO010. The outermost loop handles the next pass after a complete scan through the array and starts at SRO005.

The innermost loop at SRO010 loads HL with the entry pointed to by IY and loads DE with the next entry. A subtract is done to compare the two. If the HL entry is "heavier" than the DE entry, a swap is made by storing HL and DE and a "swap" flag in IX is set. If the HL entry is the same or "lighter," no swap occurs. The IY pointer is then incremented to point to the next entry, the count of entries in BC is decremented, and a test is made of BC. If there are more entries, a jump is made to SRO010 for the next entry comparison.

If BC is zero, all entries have been compared for this pass. IX contains the "swap" flag, and it is tested for nonzero, indicating a swap. If it is nonzero, a jump is made back to SRO005 to start over at the first entry and to reset the "swap" flag. The sort is over when a complete pass is made without the "swap" flag being set.

Sample Calling Sequence

```
NAME OF SUBROUTINE? SROARR
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
```

```
PARAMETER BLOCK VALUES?
+ 0   2   45000  LOCATION OF ARRAY
+ 2   2   5      5 ENTRIES
+ 4   0   0
MEMORY BLOCK 1 LOCATION? 45000
MEMORY BLOCK 1 VALUES?
+ 0   2   7890
+ 2   2   6789
+ 4   2   5678   INITIALIZE VALUES FOR EXAMPLE
+ 6   2   4567
+ 8   2   3456
+ 10   0   0
MEMORY BLOCK 2 LOCATION?
MOVE SUBROUTINE TO? 37777
SUBROUTINE EXECUTED AT  37777
INPUT:              OUTPUT:
HL= 40000           HL= 40000
PARAM+ 0   200      PARAM+ 0   200
PARAM+ 1   175      PARAM+ 1   175   UNCHANGED
PARAM+ 2   5        PARAM+ 2   5
PARAM+ 3   0        PARAM+ 3   0
MEMB1+ 0   210      MEMB1+ 0   128
MEMB1+ 1   30       MEMB1+ 1   13
MEMB1+ 2   133      MEMB1+ 2   215
MEMB1+ 3   26       MEMB1+ 3   17
MEMB1+ 4   46       MEMB1+ 4   46
MEMB1+ 5   22       MEMB1+ 5   22    RESORTED
MEMB1+ 6   215      MEMB1+ 6   133
MEMB1+ 7   17       MEMB1+ 7   26
MEMB1+ 8   128      MEMB1+ 8   210
MEMB1+ 9   13       MEMB1+ 9   30

NAME OF SUBROUTINE?
```

Notes

1. The bubble sort is not particularly speedy, but requires minimal memory.
2. The number of entries must be two or greater.

Program Listing

```
7F00                            ORG     7F00H           ;0522
                ;********************************************************
                ;* SORT ONE-D INTEGER ARRAY. SORTS INTEGER ARRAY INTO   *
                ;* ASCENDING ORDER.                                     *
                ;*     INPUT: HL=>PARAMETER BLOCK                       *
                ;*            PARAM+0,+1=ADDRESS OF ARRAY               *
                ;*            PARAM+2,+3=SIZE OF ARRAY                  *
                ;*     OUTPUT:ARRAY SORTED IN ASCENDING ORDER           *
                ;********************************************************
                ;
7F00 F5         SROARR  PUSH    AF              ;SAVE REGISTERS
7F01 C5                 PUSH    BC
7F02 D5                 PUSH    DE
7F03 E5                 PUSH    HL
7F04 DDE5               PUSH    IX
7F06 FDE5               PUSH    IY
7F08 CD7F0A             CALL    0A7FH           ;***GET PB LOC'N***
7F0B E5                 PUSH    HL              ;TRANSFER TO IX
7F0C DDE1               POP     IX
7F0E DD4E02     SRO005  LD      C,(IX+2)        ;PUT SIZE IN BC
7F11 DD4603             LD      B,(IX+3)
7F14 0B                 DEC     BC              ;SIZE - 1 FOR SORT
```

```
7F15 DD6E00     00320          LD      L,(IX+0)        ;PUT ADDRESS IN HL
7F18 DD6601     00330          LD      H,(IX+1)
7F1B E5         00340          PUSH    HL              ;COPY INTO IY
7F1C FDE1       00350          POP     IY
7F1E DDE5       00360          PUSH    IX              ;SAVE IX
7F20 DD210000   00370          LD      IX,0            ;SET 'NO CHANGE' FLAG
7F24 FD6E00     00380 SRO010   LD      L,(IY+0)          ;PUT CUR ENTRY INTO HL
7F27 FD6601     00390          LD      H,(IY+1)
7F2A FD5E02     00400          LD      E,(IY+2)          ;PUT NEXT ENTRY IN DE
7F2D FD5603     00410          LD      D,(IY+3)
7F30 B7         00420          OR      A                 ;CLEAR CARRY
7F31 ED52       00430          SBC     HL,DE             ;COMPARE PAIR
7F33 3811       00440          JR      C,SRO020          ;GO IF CUR<NEXT
7F35 280F       00450          JR      Z,SRO020          ;GO IF EQUAL
7F37 19         00460          ADD     HL,DE             ;RESTORE VALUE
7F38 DD23       00470          INC     IX                ;SET SWAP FLAG
7F3A FD7300     00480          LD      (IY+0),E          ;SWAP PAIR
7F3D FD7201     00490          LD      (IY+1),D
7F40 FD7502     00500          LD      (IY+2),L
7F43 FD7403     00510          LD      (IY+3),H
7F46 FD23       00520 SRO020   INC     IY                ;POINT TO NEXT ENTRY
7F48 FD23       00530          INC     IY
7F4A 0B         00540          DEC     BC                ;DECREMENT COUNT
7F4B 78         00550          LD      A,B               ;TEST COUNT
7F4C B1         00560          OR      C
7F4D 20D5       00570          JR      NZ,SRO010         ;GO IF NOT END
7F4F DDE5       00580          PUSH    IX              ;FLAG TO HL
7F51 E1         00590          POP     HL
7F52 ED42       00600          SBC     HL,BC           ;TEST FLAG
7F54 DDE1       00610          POP     IX              ;RESTORE IX
7F56 20B6       00620          JR      NZ,SRO005       ;GO IF SWAP OCCURED
7F58 FDE1       00630          POP     IY            ;RESTORE REGISTERS
7F5A DDE1       00640          POP     IX
7F5C E1         00650          POP     HL
7F5D D1         00660          POP     DE
7F5E C1         00670          POP     BC
7F5F F1         00680          POP     AF
7F60 C9         00690          RET
0000            00700          END
00000 TOTAL ERRORS
```

SROARR DECIMAL VALUES

```
245, 197, 213, 229, 221, 229, 253, 229, 205, 127,
10, 229, 221, 225, 221, 78, 2, 221, 70, 3,
11, 221, 110, 0, 221, 102, 1, 229, 253, 225,
221, 229, 221, 33, 0, 0, 253, 110, 0, 253,
102, 1, 253, 94, 2, 253, 86, 3, 183, 237,
82, 56, 17, 40, 15, 25, 221, 35, 253, 115,
0, 253, 114, 1, 253, 117, 2, 253, 116, 3,
253, 35, 253, 35, 11, 120, 177, 32, 213, 221,
229, 225, 237, 66, 221, 225, 32, 182, 253, 225,
221, 225, 225, 209, 193, 241, 201
```

CHKSUM= 242

SSNCHR: SEARCH STRING FOR N CHARACTERS

System Configuration

Model I, Model III, Model II Stand Alone.

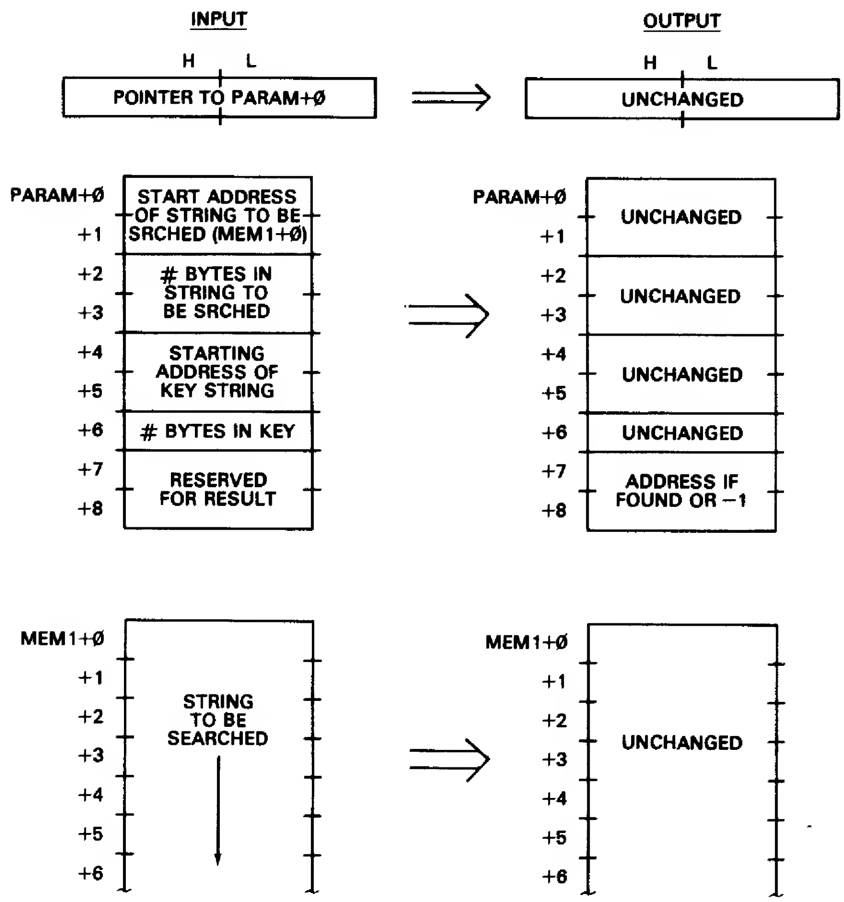
Description

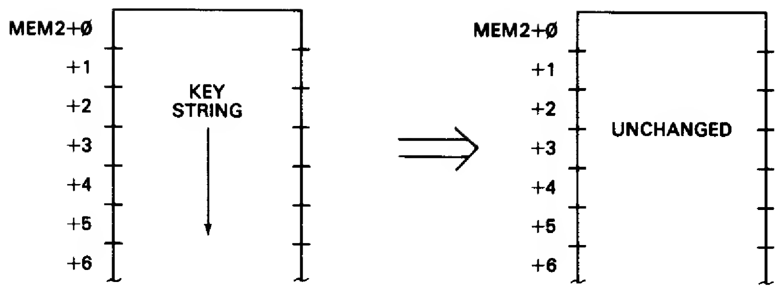
SSNCHR searches a string of any length for a "substring" of any length. A "found" or "not found" address of the substring is returned. The strings may contain any combinations of data—ASCII, binary, or other combinations.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the starting address of the string to be searched in standard Z-80 address format, least significant byte followed by most significant byte. The next two bytes of the parameter block contain the number of bytes in the string to be searched. The next two bytes of the parameter block contain the starting address of the "key" string, the string for which the search is to be made. The next two bytes in the parameter block contain the number of bytes in the key string. The next two bytes are reserved for the result.

On output, PARAM+7,+8 contain the result of the search. All other bytes in the parameter block are unchanged. The result is a −1 if the search key has not been found in the string to be searched. If the search key has been found, the result is the actual address of the first occurrence of the search key in the string to be searched.





Algorithm

The SSNCHR subroutine performs the search in two steps. First, a "CPIR" block search is made for the first character. If the first character is not found, the search has been unsuccessful. If the first character is found, a further comparison is done for the other characters in the search string.

The registers are first set up for the CPIR. The string start address of the string to be searched is put into the HL register pair. The number of bytes in the string to be searched is put into BC. The first character of the search string is put into the A register. (Also at this point, the search string start is put into DE.) The CPIR search is done at SSN060.

If the Z flag is not set after the CPIR, the first character of the string has not been found and the code at SSN080 puts a −1 into the result. If the Z flag is set, the first character of the string has been found.

The code at SSN070 compares the remaining bytes to see if the key string matches. In this loop, HL points to the locations of the string to be searched, while IY points to the locations in the key string. B contains the count of the number of characters in the key string. If any characters do not compare, a return back to the CPIR is done with HL pointing to the next byte after the byte that was found. If all characters compare, the address of the first character in the string to be searched is put into the result.

Sample Calling Sequence

```
NAME OF SUBROUTINE? SSNCHR
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0  2  45000  START OF STRING TO BE SEARCHED
+ 2  2  6      6 BYTES IN STRING TO BE SEARCHED
+ 4  2  46000  START OF KEY STRING
+ 6  1  3      3 BYTES IN KEY STRING
+ 7  2  0
+ 9  0  0
MEMORY BLOCK 1 LOCATION? 45000
MEMORY BLOCK 1 VALUES?
+ 0  1  0 ┐
+ 1  1  1 │
+ 2  1  2 ├ STRING TO BE SEARCHED
+ 3  1  3 │
+ 4  1  4 │
+ 5  1  5 ┘
+ 6  0  0
MEMORY BLOCK 2 LOCATION? 46000
```

```
MEMORY BLOCK 2 VALUES?
+ 0   1   3
+ 1   1   4   KEY STRING
+ 2   1   5
+ 3   0   0
MOVE SUBROUTINE TO? 38000
SUBROUTINE EXECUTED AT  38000
INPUT:                OUTPUT:
HL= 40000             HL= 40000
PARAM+ 0   200        PARAM+ 0   200
PARAM+ 1   175        PARAM+ 1   175
PARAM+ 2   6          PARAM+ 2   6
PARAM+ 3   0          PARAM+ 3   0     UNCHANGED
PARAM+ 4   176        PARAM+ 4   176
PARAM+ 5   179        PARAM+ 5   179
PARAM+ 6   3          PARAM+ 6   3
PARAM+ 7   0          PARAM+ 7   203   FOUND AT 45003
PARAM+ 8   0          PARAM+ 8   175
MEMB1+ 0   0          MEMB1+ 0   0
MEMB1+ 1   1          MEMB1+ 1   1
MEMB1+ 2   2          MEMB1+ 2   2
MEMB1+ 3   3          MEMB1+ 3   3
MEMB1+ 4   4          MEMB1+ 4   4     UNCHANGED
MEMB1+ 5   5          MEMB1+ 5   5
MEMB2+ 0   3          MEMB2+ 0   3
MEMB2+ 1   4          MEMB2+ 1   4
MEMB2+ 2   5          MEMB2+ 2   5

NAME OF SUBROUTINE?
```

Notes

1. The key string may be one byte.
2. The key string may not contain a larger number of bytes than the string to be searched.

Program Listing

```
7F00               00100         ORG     7F00H
                   00110 ;*****************************************************
                   00120 ;* SEARCH STRING FOR N CHARACTERS. SEARCHES STRING FOR  *
                   00130 ;* A SUBSTRING.                                         *
                   00140 ;*     INPUT: HL=> PARAMETER BLOCK                      *
                   00150 ;*            PARA,+0,+1=STARTING ADDRESS OF STRING TO  *
                   00160 ;*            BE SEARCHED                               *
                   00170 ;*            PARAM+2,+3=# BYTES IN STRING TO BE SRCHED *
                   00180 ;*            PARAM+4,+5=STARTING ADDRESS OF KEY STRING *
                   00190 ;*            PARAM+6=# OF BYTES IN KEY                 *
                   00200 ;*            PARAM+7,+8=RESERVED FOR RESULT            *
                   00210 ;*    OUTPUT:PARAM+7,+8=ADDRESS OF SUBSTRING IF FOUND    *
                   00220 ;*            OR -1 IF NOT FOUND                        *
                   00230 ;*****************************************************
                   00240 ;
7F00 F5            00250 SSNCHR  PUSH    AF              ;SAVE REGISTERS
7F01 C5            00260         PUSH    BC
7F02 D5            00270         PUSH    DE
7F03 E5            00280         PUSH    HL
7F04 DDE5          00290         PUSH    IX
7F06 FDE5          00300         PUSH    IY
7F08 CD7F0A        00310         CALL    0A7FH           ;***GET PB LOC'N***
7F0B E5            00320         PUSH    HL              ;TRANSFER TO IX
7F0C DDE1          00330         POP     IX
7F0E DD6E00        00340         LD      L,(IX+0)        ;PUT STRING START IN HL
```

```
7F11 DD6601     00350          LD      H,(IX+1)
7F14 DD4E02     00360          LD      C,(IX+2)        ;PUT # OF BYTES IN BC
7F17 DD4603     00370          LD      B,(IX+3)
7F1A DD5E04     00380          LD      E,(IX+4)        ;PUT SS IN DE
7F1D DD5605     00390          LD      D,(IX+5)
7F20 D5         00400          PUSH    DE              ;TRANSFER TO IY
7F21 FDE1       00410          POP     IY
7F23 FD7E00     00420 SSN060   LD      A,(IY+0)        ;GET FIRST CHAR OF SS
7F26 EDB1       00430          CPIR                      ;SEARCH FOR 1ST CHAR
7F28 2021       00440          JR      NZ,SSN080       ;GO IF FIRST CHAR NOT FND
7F2A DD4606     00450          LD      B,(IX+6)        ;GET # OF BYTES IN SS
7F2D 05         00460          DEC     B               ;DECREMENT FOR FIRST
7F2E 2813       00470          JR      Z,SSN072        ;ONE BYTE KEY CASE
7F30 E5         00480          PUSH    HL              ;SAVE LOC'N OF FIRST
7F31 FDE5       00490          PUSH    IY              ;SAVE 1ST CHAR OF SS
7F33 FD23       00500          INC     IY              ;POINT TO SECOND OF SS
7F35 7E         00510 SSN070   LD      A,(HL)            ;GET NEXT BYTE
7F36 FDBE00     00520          CP      (IY)              ;COMPARE
7F39 200B       00530          JR      NZ,SSN075         ;GO IF NO MATCH
7F3B 23         00540          INC     HL                ;BUMP STRING PNTR
7F3C FD23       00550          INC     IY                ;BUMP SS PNTR
7F3E 10F5       00560          DJNZ    SSN070            ;GO IF MORE
7F40 FDE1       00570          POP     IY              ;GET 1ST CHAR POS OF SS
7F42 E1         00580          POP     HL              ;RESTORE LOC'N OF FIRST+1
7F43 2B         00590 SSN072   DEC     HL              ;ADJUST FOR CPIR
7F44 1808       00600          JR      SSN090          ;GO FOR CLEANUP
7F46 FDE1       00610 SSN075   POP     IY              ;RESET
7F48 E1         00620          POP     HL              ;RESTORE CUR LOC'N
7F49 18D8       00630          JR      SSN060          ;CONTINUE CPIR
7F4B 21FFFF     00640 SSN080   LD      HL,-1           ;NOT FOUND FLAG
7F4E DD7507     00650 SSN090   LD      (IX+7),L        ;STORE LOC'N OR 'NOT FND'
7F51 DD7408     00660          LD      (IX+8),H
7F54 FDE1       00670          POP     IY              ;RESTORE REGISTERS
7F56 DDE1       00680          POP     IX
7F58 E1         00690          POP     HL
7F59 D1         00700          POP     DE
7F5A C1         00710          POP     BC
7F5B F1         00720          POP     AF
7F5C C9         00730          RET                     ;RETURN TO CALLING PROG
0000            00740          END
00000 TOTAL ERRORS
```

SSNCHR DECIMAL VALUES

```
245, 197, 213, 229, 221, 229, 253, 229, 205, 127,
10, 229, 221, 225, 221, 110, 0, 221, 102, 1,
221, 78, 2, 221, 70, 3, 221, 94, 4, 221,
86, 5, 213, 253, 225, 253, 126, 0, 237, 177,
32, 33, 221, 70, 6, 5, 40, 19, 229, 253,
229, 253, 35, 126, 253, 190, 0, 32, 11, 35,
253, 35, 16, 245, 253, 225, 225, 43, 24, 8,
253, 225, 225, 24, 216, 33, 255, 255, 221, 117,
7, 221, 116, 8, 253, 225, 221, 225, 225, 209,
193, 241, 201
```

CHKSUM= 198

SSOCHR: SEARCH STRING FOR ONE CHARACTER

System Configuration

Model I, Model III, Model II Stand Alone.

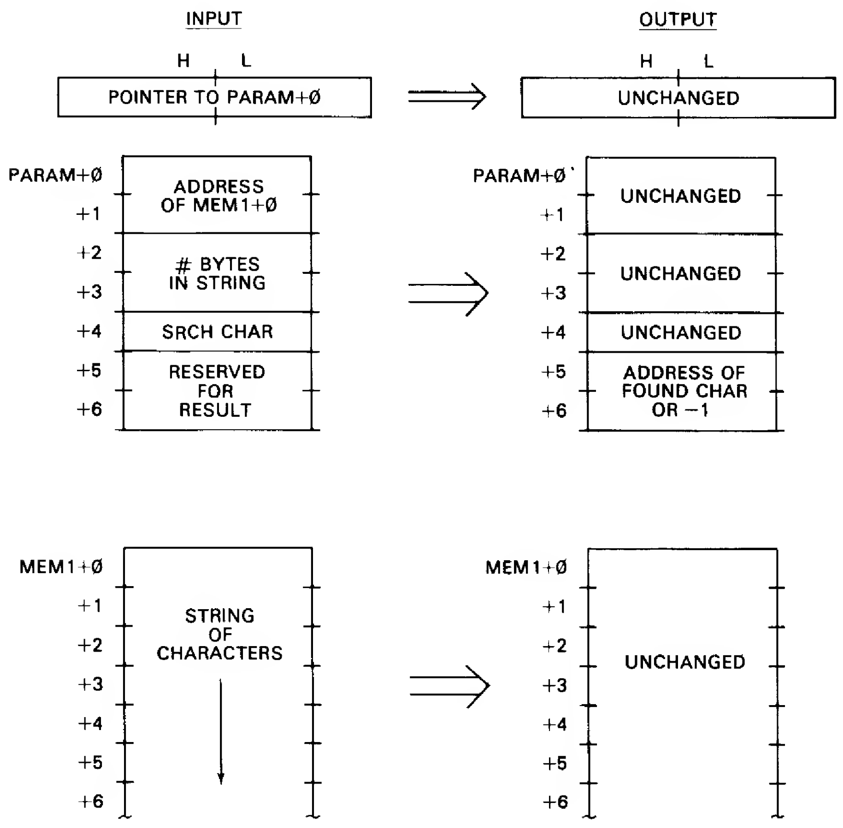
Description

SSOCHR searches a string of any length for a given byte. A "found" or "not found" address of the character is returned. The string and byte may contain any combinations of data—ASCII, binary, or other combinations.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the starting address of the string to be searched in standard Z-80 address format, least significant byte followed by most significant byte. The next two bytes of the parameter block contain the number of bytes in the string to be searched. The next bytes of the parameter block contain the "key" byte, the byte for which the search is to be made. The next two bytes are reserved for the result.

On output, PARAM+5,+6 contain the result of the search. All other bytes in the parameter block are unchanged. The result is a −1 if the search byte has not been found in the string to be searched. If the search byte has been found, the result is the actual address of the first occurrence of the search byte in the string to be searched.



Algorithm

The SSOCHR subroutine performs the search by a "CPIR" block search for the first character.

The registers are first set up for the CPIR. The string start address of the string to be searched is put into the HL register pair. The number of bytes in the string to be searched are put into BC. The search byte is put into the A register. The CPIR search is then done.

If the Z flag is not set after the CPIR, the key byte has not been found and the code at SSO010 puts a −1 into the result. If the Z flag is set, the key byte has been found.

Sample Calling Sequence

```
NAME OF SUBROUTINE? SSOCHR
HL VALUE? 50000
PARAMETER BLOCK LOCATION? 50000
PARAMETER BLOCK VALUES?
+ 0  2  40000
+ 2  2  5     ADDRESS OF STRING TO BE SEARCHED
+ 4  1  66    5 BYTES
+ 5  2  0     SEARCH CHARACTER
+ 7  0  0
MEMORY BLOCK 1 LOCATION? 40000
MEMORY BLOCK 1 VALUES?
+ 0  1  67 ┐
+ 1  1  68 │
+ 2  1  66 ├ STRING TO BE SEARCHED
+ 3  1  65 │
+ 4  1  60 ┘
+ 5  0  0
MEMORY BLOCK 2 LOCATION?
MOVE SUBROUTINE TO? 52000
SUBROUTINE EXECUTED AT  52000
INPUT:               OUTPUT:
HL= 50000            HL= 50000
PARAM+ 0  64         PARAM+ 0  64  ┐
PARAM+ 1  156        PARAM+ 1  156 │
PARAM+ 2  5          PARAM+ 2  5   ├UNCHANGED
PARAM+ 3  0          PARAM+ 3  0   │
PARAM+ 4  66         PARAM+ 4  66  ┘
PARAM+ 5  0          PARAM+ 5  66  ┐
PARAM+ 6  0          PARAM+ 6  156 ┘ FOUND AT 40002
MEMB1+ 0  67         MEMB1+ 0  67  ┐
MEMB1+ 1  68         MEMB1+ 1  68  │
MEMB1+ 2  66         MEMB1+ 2  66  ├UNCHANGED
MEMB1+ 3  65         MEMB1+ 3  65  │
MEMB1+ 4  60         MEMB1+ 4  60  ┘

NAME OF SUBROUTINE?
```

Program Listing

```
7F00         00100         ORG     7F00H           ;0522
             00110 ;*********************************************************
             00120 ;* ONE-CHARACTER STRING SEARCH. SEARCHES STRING FOR ONE *
             00130 ;* GIVEN CHARACTER.                                      *
             00140 ;*     INPUT: HL=> PARAMETER BLOCK                        *
             00150 ;*            PARAM+0,+1=ADDRESS OF STRING TO BE SRCHED   *
             00160 ;*            PARAM+2,+3=# OF BYTES                       *
             00170 ;*            PARAM+4=SEARCH CHARACTER                    *
             00180 ;*            PARAM+5,+6=RESERVED FOR RESULT              *
             00190 ;*     OUTPUT:PARAM+5,+6 SET TO -1 IF NOT FOUND OR ADD-   *
             00200 ;*            RESS OF CHARACTER IF FOUND                  *
             00210 ;*********************************************************
             00220 ;
```

```
7F00 F5         00230 SSOCHR  PUSH    AF              ;SAVE REGISTERS
7F01 C5         00240         PUSH    BC
7F02 E5         00250         PUSH    HL
7F03 DDE5       00260         PUSH    IX
7F05 CD7F0A     00270         CALL    0A7FH           ;***GET PB LOC'N***
7F08 E5         00280         PUSH    HL              ;TRANSFER TO IX
7F09 DDE1       00290         POP     IX
7F0B DD6E00     00300         LD      L,(IX+0)        ;PUT STRING ADDRESS IN HL
7F0E DD6601     00310         LD      H,(IX+1)
7F11 DD4E02     00320         LD      C,(IX+2)        ;PUT # BYTES IN BC
7F14 DD4603     00330         LD      B,(IX+3)
7F17 DD7E04     00340         LD      A,(IX+4)        ;PUT SEARCH KEY IN A
7F1A EDB1       00350         CPIR                     ;SEARCH
7F1C 2003       00360         JR      NZ,SS0010       ;GO IF NOT FOUND
7F1E 2B         00370         DEC     HL              ;FOUND, ADJUST POINTER
7F1F 1803       00380         JR      SS0020          ;GO TO STORE RESULT
7F21 21FFFF     00390 SS0010  LD      HL,-1           ;FLAG FOR NOT FOUND
7F24 DD7505     00400 SS0020  LD      (IX+5),L        ;STORE RESULT
7F27 DD7406     00410         LD      (IX+6),H
7F2A DDE1       00420         POP     IX              ;RESTORE REGISTERS
7F2C E1         00430         POP     HL
7F2D C1         00440         POP     BC
7F2E F1         00450         POP     AF
7F2F C9         00460         RET                     ;RETURN TO CALLING PROG
0000            00470         END
00000 TOTAL ERRORS
```

```
SSOCHR DECIMAL VALUES

245, 197, 229, 221, 229, 205, 127, 10, 229, 221,
225, 221, 110, 0, 221, 102, 1, 221, 78, 2,
221, 70, 3, 221, 126, 4, 237, 177, 32, 3,
43, 24, 3, 33, 255, 255, 221, 117, 5, 221,
116, 6, 221, 225, 225, 193, 241, 201

CHKSUM= 137
```

SSTCHR: SEARCH STRING FOR TWO CHARACTERS

System Configuration

Model I, Model III, Model II Stand Alone.

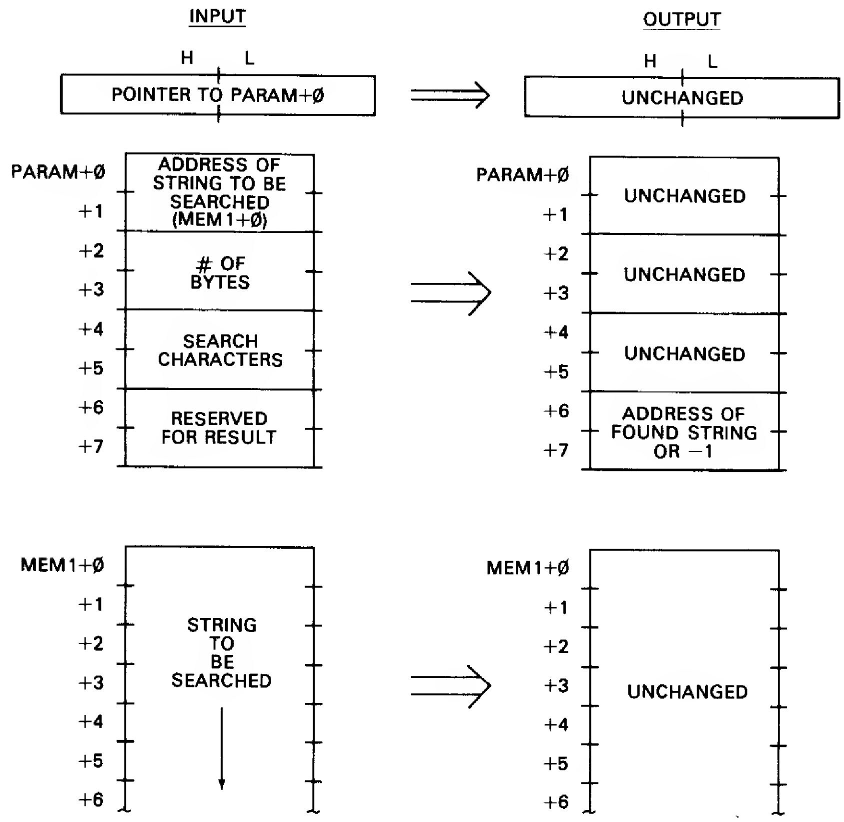
Description

SSTCHR searches a string of any length for a "substring" of two bytes. A "found" or "not found" address of the substring is returned. The strings may contain any combinations of data—ASCII, binary, or other combinations.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain the starting address of the string to be searched in standard Z-80 address format, least significant byte followed by most significant byte. The next two bytes of the parameter block contain the number of bytes in the string to be searched. The next two bytes of the parameter block contain the "key" string, the string for which the search is to be made. The next two bytes are reserved for the result.

On output, PARAM+6,+7 contain the result of the search. All other bytes in the parameter block are unchanged. The result is a −1 if the search key has not been found in the string to be searched. If the search key has been found, the result is the actual address of the first occurrence of the search key in the string to be searched.



Algorithm

The SSTCHR subroutine performs the search in two steps. First, a "CPIR" block search is made for the first character. If the first character is not found, the search has been unsuccessful. If the first character is found, a further comparison is done for the second character in the search string.

The registers are first set up for the CPIR. The string start address of the string to be searched is put into the HL register pair. The number of bytes in the string to be searched is put into BC. The first character of the search string is put into the A register. The CPIR search is then done.

If the Z flag is not set after the CPIR, the first character of the string has not been found and the code at SST020 puts a −1 into the result. If the Z flag is set, the first character of the string has been found.

The code following the CPIR compares the remaining byte to see if the key string matches. In this loop, HL points to the location of the second byte in the string to be searched, while IX points to the parameter block location. If the second character does not compare; a return back to the CPIR is done with HL pointing to the next byte after the byte that was found. If the second character compares, the address of the first character in the string to be searched is put into the result.

Sample Calling Sequence

```
NAME OF SUBROUTINE? SSTCHR
HL VALUE? 42222
PARAMETER BLOCK LOCATION? 42222
PARAMETER BLOCK VALUES?
+ 0   2   45555  START OF STRING TO BE SEARCHED
+ 2   2   7      7 BYTES IN STRING TO BE SEARCHED
+ 4   1   49  ]-SEARCH CHARACTERS
+ 5   1   48  ]
+ 6   2   0
+ 8   0   0
MEMORY BLOCK 1 LOCATION? 45555
MEMORY BLOCK 1 VALUES?
+ 0   1   45  ]
+ 1   1   46  ]
+ 2   1   47  ]
+ 3   1   48  ]- INITIALIZE STRING TO BE SEARCHED
+ 4   1   49  ]  FOR EXAMPLE
+ 5   1   48  ]
+ 6   1   47  ]
+ 7   0   0
MEMORY BLOCK 2 LOCATION?
MOVE SUBROUTINE TO? 38000
SUBROUTINE EXECUTED AT  38000
INPUT:              OUTPUT:
HL= 42222           HL= 42222
PARAM+ 0   243      PARAM+ 0   243  ]
PARAM+ 1   177      PARAM+ 1   177  ]
PARAM+ 2   7        PARAM+ 2   7    ]-UNCHANGED
PARAM+ 3   0        PARAM+ 3   0    ]
PARAM+ 4   49       PARAM+ 4   49   ]
PARAM+ 5   48       PARAM+ 5   48   ]
PARAM+ 6   0        PARAM+ 6   247  ]- FOUND AT 45559
PARAM+ 7   0        PARAM+ 7   177  ]
MEMB1+ 0   45       MEMB1+ 0   45   ]
MEMB1+ 1   46       MEMB1+ 1   46   ]
MEMB1+ 2   47       MEMB1+ 2   47   ]
MEMB1+ 3   48       MEMB1+ 3   48   ]-UNCHANGED
MEMB1+ 4   49       MEMB1+ 4   49   ]
MEMB1+ 5   48       MEMB1+ 5   48   ]
MEMB1+ 6   47       MEMB1+ 6   47   ]

NAME OF SUBROUTINE?
```

Notes

1. If a search is to be made for an address, the order of the search key should be least significant byte followed by most significant byte. If the search is for character data, the order of the search key should be first character, second character. In other words, arrange the bytes the way they would occur in the string to be searched.

Program Listing

```
7F00              ORG     7F00H           ;0522
      ;*********************************************************
      ;* TWO-CHARACTER STRING SEARCH. SEARCHES STRING FOR TWO *
      ;* GIVEN CHARACTERS.                                     *
      ;*    INPUT: HL=> PARAMETER BLOCK                        *
      ;*           PARAM+0,+1=ADDRESS OF STRING TO BE SRCHED   *
      ;*           PARAM+2,+3=# OF BYTES                       *
      ;*           PARAM+4,+5=SEARCH CHARACTERS                *
      ;*           PARAM+6,+7=RESERVED FOR RESULT              *
      ;*    OUTPUT:PARAM+6,+7 SET TO -1 IF NOT FOUND OR ADD-   *
      ;*           RESS OF CHARACTERS IF FOUND                 *
      ;*********************************************************
```

```
                 00220 ;
7F00 F5          00230 SSTCHR  PUSH   AF              ;SAVE REGISTERS
7F01 C5          00240         PUSH   BC
7F02 E5          00250         PUSH   HL
7F03 DDE5        00260         PUSH   IX
7F05 CD7F0A      00270         CALL   0A7FH           ;***GET PB LOC'N***
7F08 E5          00280         PUSH   HL              ;TRANSFER TO IX
7F09 DDE1        00290         POP    IX
7F0B DD6E00      00300         LD     L,(IX+0)        ;PUT STRING ADDRESS IN HL
7F0E DD6601      00310         LD     H,(IX+1)
7F11 DD4E02      00320         LD     C,(IX+2)        ;PUT # BYTES IN BC
7F14 DD4603      00330         LD     B,(IX+3)
7F17 DD7E04      00340 SST010  LD     A,(IX+4)          ;PUT SEARCH KEY IN A
7F1A EDB1        00350         CPIR                       ;SEARCH
7F1C 200D        00360         JR     NZ,SST020         ;GO IF NOT FOUND
7F1E 78          00370         LD     A,B               ;TEST FOR END
7F1F B1          00380         OR     C
7F20 2809        00390         JR     Z,SST020          ;GO IF AT END OF STRING
7F22 DD7E05      00400         LD     A,(IX+5)          ;GET SECOND CHAR OF KEY
7F25 BE          00410         CP     (HL)              ;COMPARE TO NEXT BYTE
7F26 20EF        00420         JR     NZ,SST010         ;CONTINUE IF NO MATCH
7F28 2B          00430         DEC    HL              ;ADJUST BACK TO START
7F29 1803        00440         JR     SST030          ;GO TO STORE RESULT
7F2B 21FFFF      00450 SST020  LD     HL,-1           ;FLAG FOR NOT FOUND
7F2E DD7506      00460 SST030  LD     (IX+6),L        ;STORE RESULT
7F31 DD7407      00470         LD     (IX+7),H
7F34 DDE1        00480         POP    IX              ;RESTORE REGISTERS
7F36 E1          00490         POP    HL
7F37 C1          00500         POP    BC
7F38 F1          00510         POP    AF
7F39 C9          00520         RET                    ;RETURN TO CALLING PROG
0000             00530         END
00000 TOTAL ERRORS
```

SSTCHR DECIMAL VALUES

```
245, 197, 229, 221, 229, 205, 127, 10, 229, 221,
225, 221, 110, 0, 221, 102, 1, 221, 78, 2,
221, 70, 3, 221, 126, 4, 237, 177, 32, 13,
120, 177, 40, 9, 221, 126, 5, 190, 32, 239,
43, 24, 3, 33, 255, 255, 221, 117, 6, 221,
116, 7, 221, 225, 225, 193, 241, 201
```

CHKSUM= 28

SXCASS: WRITE/READ SCREEN CONTENTS TO CASSETTE

System Configuration

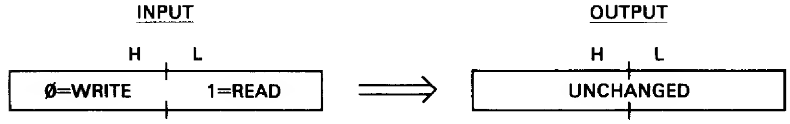
Model I, Model III.

Description

SXCASS writes the video display as a cassette record or reads in a previously written record to the display. All screen characters and graphics are written to the cassette and the subsequent read will restore the entire screen as it appeared before the write.

Input/Output Parameters

On input, the HL register pair contains a zero for a write or a one for a read. On output, the screen has been written as a single cassette record, or the next cassette record has been read to the screen.



Algorithm

If a screen write is to be performed, the code at SXC010 is executed. This uses the ROM subroutine to write leader (287H) of zeroes and a sync byte. The loop at SXC010 calls the ROM "write cassette byte" subroutine to write the video display memory contents from location 3C00H through 3FFFH. HL contains the pointer to video display memory. The write is done until the H register contains 40H, signifying that the last screen byte has been written. No checksum or other header data is put on the cassette record.

If a read screen is to be performed, the code at SXC025 is executed. ROM subroutine 296H is called to bypass the leader of the next cassette record. The loop at SXC030 calls the ROM "read cassette byte" subroutine to read in the bytes of the next cassette record into video memory locations 3C00H through 3FFFH. HL is used as a memory pointer. The read is done until the H register contains 40H, signifying that the last screen byte has been read.

Sample Calling Sequence

```
NAME OF SUBROUTINE? SXCASS
HL VALUE? 0    WRITE
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 37777
SUBROUTINE EXECUTED AT  37777
INPUT:            OUTPUT:
HL= 0             HL= 0

NAME OF SUBROUTINE?
```

Notes

1. The read or write operation takes approximately 25 seconds.
2. This subroutine does not save registers.

Program Listing

```
7F00                  ORG     7F00H           ;0520
          ;**********************************************************
          ;* WRITE/READ SCREEN CONTENTS TO CASSETTE.                *
          ;*     INPUT: HL=0 FOR WRITE SCREEN, 1 FOR READ           *
          ;*     OUTPUT: SCREEN/CASSETTE ACTIONS                    *
          ;**********************************************************
          ;
```

```
7F00 F3       00170 SXCASS  DI                    ;DISABLE INTERRUPTS
7F01 AF       00180         XOR    A              ;ZERO A
7F02 CD1202   00190         CALL   212H           ;SELECT CASSETTE 0
7F05 CD7F0A   00200         CALL   0A7FH          ;***GET FUNCTION***
7F08 CB45     00210         BIT    0,L            ;TEST FUNCTION
7F0A 2014     00220         JR     NZ,SXC025      ;GO IF READ CASSETTE
              00230 ; WRITE HERE
7F0C CD8702   00240         CALL   287H           ;WRITE LEADER
7F0F 21003C   00250         LD     HL,3C00H       ;START OF SCREEN
7F12 E5       00260 SXC010  PUSH   HL               ;SAVE CURRENT LOCATION
7F13 7E       00270         LD     A,(HL)           ;GET NEXT BYTE
7F14 CD6402   00280         CALL   264H             ;WRITE TO CASSETTE
7F17 E1       00290         POP    HL               ;RESTORE POINTER
7F18 23       00300         INC    HL               ;BUMP POINTER
7F19 7C       00310         LD     A,H              ;GET POINTER MSB
7F1A FE40     00320         CP     40H              ;TEST FOR SCREEN END+1
7F1C 20F4     00330         JR     NZ,SXC010        ;LOOP IF NOT END
7F1E 1812     00340         JR     SXC040         ;CLEANUP
              00350 ; READ HERE
7F20 CD9602   00360 SXC025  CALL   296H           ;BYPASS LEADER
7F23 21003C   00370         LD     HL,3C00H       ;START OF SCREEN
7F26 E5       00380 SXC030  PUSH   HL               ;SAVE CURRENT LOCATION
7F27 CD3502   00390         CALL   235H             ;READ NEXT BYTE
7F2A E1       00400         POP    HL               ;RESTORE POINTER
7F2B 77       00410         LD     (HL),A           ;STORE BYTE
7F2C 23       00420         INC    HL               ;BUMP POINTER
7F2D 7C       00430         LD     A,H              ;GET POINTER MSB
7F2E FE40     00440         CP     40H              ;TEST FOR SCREEN END+1
7F30 20F4     00450         JR     NZ,SXC030        ;LOOP IF NOT END
7F32 CDF801   00460 SXC040  CALL   1F8H           ;DESELECT
7F35 C9       00470         RET                   ;RETURN TO CALLING PROG
0000          00480         END
00000 TOTAL ERRORS
```

```
SXCASS DECIMAL VALUES

243, 175, 205, 18, 2, 205, 127, 10, 203, 69,
32, 20, 205, 135, 2, 33, 0, 60, 229, 126,
205, 100, 2, 225, 35, 124, 254, 64, 32, 244,
24, 18, 205, 150, 2, 33, 0, 60, 229, 205,
53, 2, 225, 119, 35, 124, 254, 64, 32, 244,
205, 248, 1, 201

CHKSUM= 229
```

TIMEDL: TIME DELAY

System Configuration

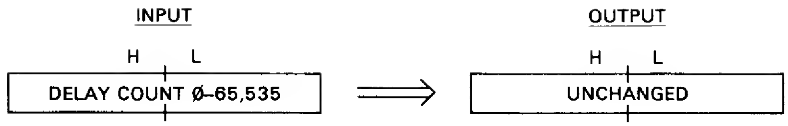
Model I, Model III, Model II Stand Alone.

Description

TIMEDL delays a specified amount of time, from 1 millisecond to 65,536 milliseconds, before returning to the user calling program.

Input/Output Parameters

On input, the HL register pair contains the number of milliseconds to delay, from 1 to 65,536. A value of zero is treated as 65,536. TIMEDL returns after the specified delay.



Algorithm

The 1 millisecond time delay loop is the heart of TIMEDL. It consists of one instruction, the DJNZ at TIM020. This instruction takes 13 cycles when the loop is made or 8 cycles when B is decremented to zero. With a given count in B, therefore, the time delay is:

Delay (cycles) = (CNT−1)*13 + 8

A cycle in the Model I with a standard clock takes 0.56375 microseconds. The delay in microseconds is therefore:

Delay (microseconds) = (CNT−1)*7.32875 + 4.51

To get a time delay of 1000 microseconds (1 millisecond):

1000 = (CNT−1)*7.32875 + 4.51;
CNT= 134.83

The outer loop of TIMEDL controls the number of 1 millisecond inner loops. The outer loop has some overhead associated with it, so the count in B for the DJNZ is made 134 even. The actual time delay for a given value in HL, HLCNT, is now:

Delay (cycles) = HLCNT*(7 + (133*13+8) +15 +12)
Delay (microseconds) = HLCNT*998.40

This is about a 0.1% error on the low side, or about a millisecond for a one-second delay.

Sample Calling Sequence

```
NAME OF SUBROUTINE? TIMEDL
HL VALUE? Ø    MAXIMUM DELAY = 65.535 SECONDS
PARAMETER BLOCK LOCATION?
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 5ØØØØ
SUBROUTINE EXECUTED AT  5ØØØØ
INPUT:              OUTPUT:
HL= Ø               HL= Ø

NAME OF SUBROUTINE?
```

Notes

1. Adjust the immediate value loaded into B for clock modified TRS-80s.
2. Use an immediate value of 153 for Model IIIs.
3. Use an immediate value of 151 for Model IIs for delays of .5 to 32768 milliseconds in units of 1/2 millisecond.

```
7F00                       ORG     7F00H           ;0520
                   ;*****************************************************
                   ; TIME DELAY. DELAYS 1 TO 65,536 MILLISECONDS.        *
                   ;     INPUT: HL=TIME DELAY COUNT, 1 TO 65535. 0=65536 *
                   ;     OUTPUT:RETURN AFTER DELAY                       *
                   ;*****************************************************
                   ;
7F00 C5            TIMEDL  PUSH    BC              ;SAVE REGISTERS
7F01 D5                    PUSH    DE
7F02 E5                    PUSH    HL
7F03 CD7F0A                CALL    0A7FH           ;***GET TD COUNT***
7F06 110100                LD      DE,1            ;DECREMENT
7F09 0686          TIM010  LD      B,134             ;INNER LOOP COUNT 7
7F0B 10FE          TIM020  DJNZ    TIM020              ;LOOP FOR 1 MS 8/13
7F0D ED52                  SBC     HL,DE              ;DECREMENT TD COUNT 15
7F0F 20F8                  JR      NZ,TIM010          ;GO IF NOT OVER 7/12
7F11 E1                    POP     HL              ;RESTORE REGISTERS
7F12 D1                    POP     DE
7F13 C1                    POP     BC
7F14 C9                    RET                     ;RETURN TO CALLING PROG
0000                       END
00000 TOTAL ERRORS
```

```
TIMEDL DECIMAL VALUES


 197, 213, 229, 205, 127, 10, 17, 1, 0, 6,
 134, 16, 254, 237, 82, 32, 248, 225, 209, 193,
 201

CHKSUM= 20
```

TONOUT: TONE ROUTINE

System Configuration

Model I, Model III.

Description

TONOUT outputs a tone through the cassette port. The cassette jack output may be connected to a small, inexpensive amplifier for audio sound effects or warning tones. The tone ranges from approximately 0 cycles per second (hertz) to 14,200 cycles per second. The duration of the tone may be specified by the user.

TONOUT is not a musical tone generator (see MUNOTE), but is a general-purpose tone generator to produce tones over a wide range and duration.

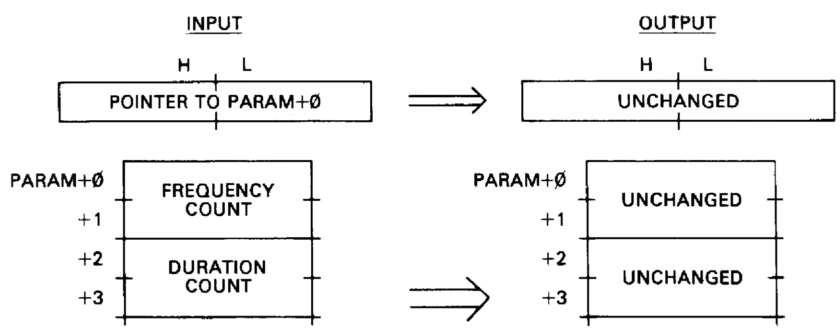
Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block contain a frequency count for the subroutine. The frequency count may be 1 to 65,535. A frequency count of 0 is regarded as

65,536. The frequency decreases as the frequency count increases. A frequency count of 1 is approximately 14,200 hertz, while a frequency count of 256 is approximately 150 hertz. The exact frequency is given by

Frequency = 1,000,000 / (25.9*COUNT + 44.53)

The next two bytes of the parameter block contain a duration count of 1 to 65,535. A duration count of 0 is regarded as 65,536. The greater the duration count, the greater will be the duration of the tone. Each duration count produces one "cycle" of the tone plus one additional cycle. A tone of 400 hertz, for example, is 1/400 or 2.5 milliseconds per cycle, and a duration count of 100 would cause the 400 hertz tone to be generated for 100*2.5 milliseconds or 1/4 second. The higher the frequency, the smaller the cycle time, and the duration count should be adjusted to compensate for this. Two consecutive 400 hertz and 800 hertz tones of 1/4-second duration, for example, should have duration counts of 100 and 50, respectively. Maximum duration for a 1000 hertz tone is 65.5 seconds.



Algorithm

TONOUT uses two loops. The outer loop (from TON010) produces the number of cycles equal to the duration count. The inner loop is made up of two parts. The TON020 portion outputs an "on" pulse from the cassette output. The TON030 portion turns off the cassette port for the same period of time. Both portions use the frequency count from the parameter block for a timing loop count.

The frequency count is first put into DE and the duration count into IX. The TON010 loop puts the DE frequency count into HL and turns on the cassette (OUT 0FFH,A). The count in HL is then decremented by one in the TON020 timing loop. At the end of the loop, the count is again put into HL from DE, the cassette is turned off, and the count is decremented by one in the TON030 timing loop. After this loop, the duration, or cycle, count in IX is decremented by one and if not negative, a jump is made back to TON010 for the next cycle.

Sample Calling Sequence

```
NAME OF SUBROUTINE? TONOUT
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
```

```
+ 0   2   37       FREQUENCY COUNT OF ABOUT 1000 HZ
+ 2   2   10000    DURATION OF ABOUT 10 SECONDS
+ 4   0   0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 37000
SUBROUTINE EXECUTED AT  37000
INPUT:                OUTPUT:
HL= 40000             HL= 40000
PARAM+ 0   37         PARAM+ 0   37
PARAM+ 1   0          PARAM+ 1   0    UNCHANGED
PARAM+ 2   16         PARAM+ 2   16
PARAM+ 3   39         PARAM+ 3   39

NAME OF SUBROUTINE?
```

Notes

1. Cassette port electronics limits the tone output to 100 through 6000 hertz or so.

2. The frequency equation above is for a standard TRS-80 Model I clock frequency.

Program Listing

```
7F00                       ORG     7F00H              ;0522
                   ;*****************************************************
                   ;* TONE ROUTINE. OUTPUTS A TONE THROUGH THE CASSETTE  *
                   ;* PORT OF SPECIFIED FREQUENCY AND DURATION.           *
                   ;*    INPUT: HL=> PARAMETER BLOCK                      *
                   ;*              PARAM+0,+1=FREQUENCY COUNT             *
                   ;*              PARAM+2,+3=DURATION COUNT              *
                   ;*    OUTPUT:TONE ON CASSETTE PORT                     *
                   ;*****************************************************
                   ;
7F00 F5            TONOUT  PUSH    AF                 ;SAVE REGISTERS
7F01 C5                    PUSH    BC
7F02 D5                    PUSH    DE
7F03 E5                    PUSH    HL
7F04 DDE5                  PUSH    IX
7F06 CD7F0A                CALL    0A7FH              ;***GET PB LOC'N***
7F09 E5                    PUSH    HL                 ;TRANSFER TO IX
7F0A DDE1                  POP     IX
7F0C DD5E00                LD      E,(IX+0)           ;PUT FREQ COUNT IN DE
7F0F DD5601                LD      D,(IX+1)
7F12 1B                    DEC     DE                 ;ADJUST FOR LOOP
7F13 DD4E02                LD      C,(IX+2)           ;PUT DUR COUNT IN BC
7F16 DD4603                LD      B,(IX+3)
7F19 0B                    DEC     BC                 ;ADJUST FOR LOOP
7F1A C5                    PUSH    BC                 ;TRANSFER TO IX
7F1B DDE1                  POP     IX
7F1D 01FFFF                LD      BC,-1              ;FOR TIGHT LOOP
7F20 6B            TON010  LD      L,E                  ;PUT FREQ COUNT IN HL 4
7F21 62                    LD      H,D                  ;4
7F22 3E01                  LD      A,1                  ;MAXIMUM POSITIVE 7
7F24 D3FF                  OUT     (0FFH),A             ;OUTPUT 11
7F26 09            TON020  ADD     HL,BC                  ;COUNT-1 11
7F27 DA267F                JP      C,TON020               ;LP FOR 1/2 CYC 7/12
7F2A 6B                    LD      L,E                  ;PUT FREQ COUNT IN HL 4
7F2B 62                    LD      H,D                  ;4
7F2C 3E02                  LD      A,2                  ;MAXIMUM NEGATIVE 7
7F2E D3FF                  OUT     (0FFH),A             ;OUTPUT 11
7F30 09            TON030  ADD     HL,BC                  ;COUNT-1 11
```

```
7F31 38FD       00480          JR      C,TON030           ;LP FOR 1/2 CYC 7/12
7F33 DD09       00490          ADD     IX,BC             ;DECREMENT DUR COUNT 15
7F35 DA207F     00500          JP      C,TON010          ;LOOP IF NOT DONE 7/12
7F38 DDE1       00510          POP     IX               ;RESTORE REGISTERS
7F3A E1         00520          POP     HL
7F3B D1         00530          POP     DE
7F3C C1         00540          POP     BC
7F3D F1         00550          POP     AF
7F3E C9         00560          RET                      ;RETURN TO CALLING PROG
0000            00570          END
00000 TOTAL ERRORS
```

```
TONOUT DECIMAL VALUES


 245, 197, 213, 229, 221, 229, 205, 127, 10, 229,
 221, 225, 221, 94, 0, 221, 86, 1, 27, 221,
 78, 2, 221, 70, 3, 11, 197, 221, 225, 1,
 255, 255, 107, 98, 62, 1, 211, 255, 9, 218,
 38, 127, 107, 98, 62, 2, 211, 255, 9, 56,
 253, 221, 9, 218, 32, 127, 221, 225, 225, 209,
 193, 241, 201

CHKSUM= 102
```

WCRECD: WRITE RECORD TO CASSETTE

System Configuration

Model I, Model III.

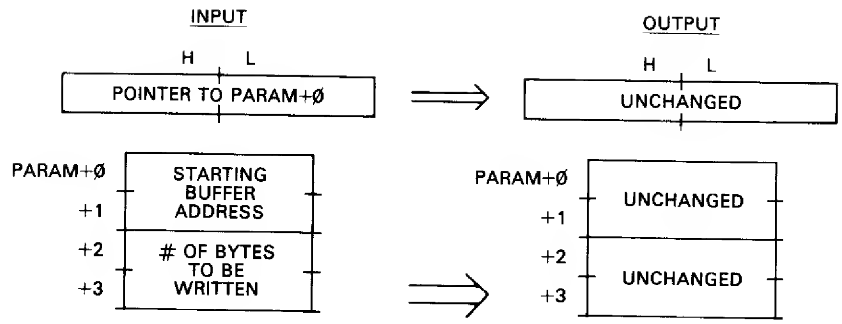
Description

WCRECD writes a variable-length record from memory to cassette. The record may be any number of bytes, from 1 to the limits of memory. The record is prefixed by a four-byte header that holds the starting address and number of bytes in the remainder of the record. The record is terminated by a checksum byte that is the additive checksum of all bytes in the record. Data in memory may represent any type of data the user desires; the record is written out as a "core image."

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first two bytes of the parameter block are the starting address of the data to be written out, in standard Z-80 address format, least significant byte followed by most significant byte. The next two bytes of the parameter block are the number of bytes to be written in the record, 1 to 65,535. A value of 0 is treated as 65,536 bytes.

On output, the contents of the parameter block are unchanged and the record has been written to cassette.



Algorithm

The WCRECD subroutine uses Level II or Level III ROM subroutines to perform the write. First, a CALL is made to 212H to select cassette 0. Next, a call is made to 287H to write 256 zeroes and a sync byte as leader for the cassette record.

The four-byte header is written out in the WCR005 loop. This header is taken from the parameter block and consists of the two address bytes and the two bytes containing the number of bytes in the record. Each byte is written by a CALL to 264H. A checksum in B is cleared before the operation; after the four-byte write, it contains the partial checksum for the four bytes.

The starting address for the data and the number of bytes is next put into HL and DE, respectively. The loop at WCR010 writes out all of the bytes in the memory block by CALLS to 264H. For each CALL, the current value of the byte is added to the B checksum subtotal, the pointer to memory in HL is bumped by one, and the count in DE is decremented by one. When DE reaches zero, the checksum in B is output as the last byte and the cassette is deselected by a CALL to 1F8H.

Sample Calling Sequence

```
NAME OF SUBROUTINE? WCRECD
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0  2  15360   BUFFER
+ 2  2  1024    1024 BYTES
+ 4  0  0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 38000
SUBROUTINE EXECUTED AT  38000
INPUT:              OUTPUT:
HL= 40000           HL= 40000
PARAM+ 0  0         PARAM+ 0  0  ┐
PARAM+ 1  60        PARAM+ 1  60 ├UNCHANGED
PARAM+ 2  0         PARAM+ 2  0  │
PARAM+ 3  4         PARAM+ 3  4  ┘

NAME OF SUBROUTINE?
```

Notes

1. This subroutine uses cassette 0 only.
2. For 500 baud tape operations, each 1000 bytes will take about 20 seconds.
3. This subroutine does not save registers.

```
7F00                     ORG     7F00H           ;0520
                 ;**************************************************
                 ;* WRITE RECORD TO CASSETTE. WRITES A VARIABLE-LENGTH   *
                 ;* RECORD TO CASSETTE FROM A GIVEN BUFFER.              *
                 ;*     INPUT: HL=> PARAMETER BLOCK                      *
                 ;*            PARAM+0,+1=STARTING BUFFER ADDRESS        *
                 ;*            PARAM+2,+3=NUMBER OF BYTES TO BE WRITTEN  *
                 ;*     OUTPUT:RECORD WRITTEN TO CASSETTE                *
                 ;**************************************************
                 ;
7F00 F3          WCRECD  DI                      ;DISABLE INTERRUPTS
7F01 AF                  XOR     A               ;ZERO A
7F02 CD1202              CALL    212H            ;SELECT CASSETTE 0
7F05 CD8702              CALL    287H            ;WRITE LEADER
7F08 CD7F0A              CALL    0A7FH           ;***GET PAR BL ADDR***
7F0B E5                  PUSH    HL              ;SAVE
7F0C 010004              LD      BC,1024+0       ;4 TO B, 0 TO C
7F0F 7E          WCR005  LD      A,(HL)            ;GET HEADER BYTE
7F10 F5                  PUSH    AF                ;SAVE BYTE
7F11 81                  ADD     A,C               ;CHECKSUM
7F12 4F                  LD      C,A               ;SAVE CHECKSUM
7F13 F1                  POP     AF                ;RESTORE ORIG BYTE
7F14 C5                  PUSH    BC                ;SAVE COUNT, CHECKSUM
7F15 E5                  PUSH    HL                ;SAVE POINTER
7F16 CD6402              CALL    264H              ;WRITE BYTE TO CASSETTE
7F19 E1                  POP     HL                ;RESTORE POINTER
7F1A C1                  POP     BC                ;GET COUNT, CHECKSUM
7F1B 23                  INC     HL                ;BUMP POINTER
7F1C 10F1                DJNZ    WCR005            ;LOOP FOR 4 HEADER BYTES
7F1E DDE1                POP     IX              ;COMPLETE TRANSFER TO IX
7F20 41                  LD      B,C             ;CHECKSUM
7F21 DD6E00              LD      L,(IX+0)        ;GET STARTING ADDRESS
7F24 DD6601              LD      H,(IX+1)
7F27 DD5E02              LD      E,(IX+2)        ;GET # BYTES
7F2A DD5603              LD      D,(IX+3)
7F2D C5          WCR010  PUSH    BC                ;SAVE CHECKSUM
7F2E D5                  PUSH    DE                ;SAVE # OF BYTES
7F2F E5                  PUSH    HL                ;SAVE CURENT LOCATION
7F30 7E                  LD      A,(HL)            ;GET NEXT BYTE
7F31 CD6402              CALL    264H              ;WRITE TO CASSETTE
7F34 E1                  POP     HL                ;RESTORE POINTER
7F35 D1                  POP     DE                ;RESTORE # OF BYTES
7F36 C1                  POP     BC                ;GET CHECKSUM
7F37 7E                  LD      A,(HL)            ;BYTE JUST OUTPUT
7F38 80                  ADD     A,B               ;COMPUTE CHECKSUM
7F39 47                  LD      B,A               ;SAVE
7F3A 23                  INC     HL                ;BUMP POINTER
7F3B 1B                  DEC     DE                ;DECREMENT # BYTES
7F3C 7A                  LD      A,D               ;TEST FOR ZERO
7F3D B3                  OR      E
7F3E 20ED                JR      NZ,WCR010         ;LOOP IF NOT END
7F40 78                  LD      A,B             ;GET CHECKSUM
7F41 CD6402              CALL    264H            ;OUTPUT AS LAST BYTE
7F44 CDF801              CALL    1F8H            ;DESELECT
7F47 C9                  RET                     ;RETURN TO CALLING PROG
0000                     END
```

WCRECD DECIMAL VALUES

```
243, 175, 205, 18, 2, 205, 135, 2, 205, 127,
10, 229, 1, 0, 4, 126, 245, 129, 79, 241,
197, 229, 205, 100, 2, 225, 193, 35, 16, 241,
221, 225, 65, 221, 110, 0, 221, 102, 1, 221,
```

```
94, 2, 221, 86, 3, 197, 213, 229, 126, 205,
100, 2, 225, 209, 193, 126, 128, 71, 35, 27,
122, 179, 32, 237, 120, 205, 100, 2, 205, 248,
1, 201
```

```
CHKSUM= 139
```

WRDSEC: WRITE DISK SECTOR

System Configuration

Model I.

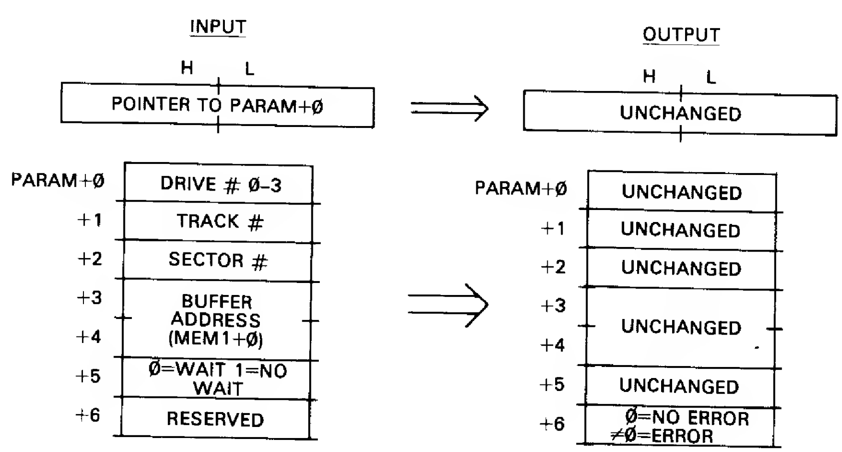
Description

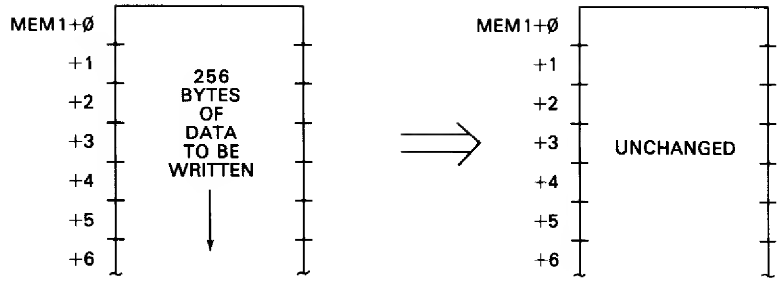
WRDSEC writes one sector from a specified buffer area to a specified disk drive. The user must know where a particular file is to be and what sectors are involved to utilize this subroutine. It is not a general-purpose "file manage" subroutine.

Input/Output Parameters

On input, the HL register pair contains a pointer to a parameter block. The first byte of the parameter block contains the disk drive number, 0 to 3, corresponding to disk drives 1 through 4. The next byte of the parameter block contains the track number, 0 through N. (Standard TRS-80s use disk drives with 35 tracks; other drives are available for 40 tracks.) The next byte is the sector number, 0 through N (0 through 9 will be the most common range). The next two bytes are the user buffer area for the write in standard Z-80 address format, least significant byte followed by most significant byte. The next byte contains a zero if a wait is to occur until the disk drive motor is brought up to speed; the byte contains a 1 if the motor is running (disk operation has just been completed) and no wait is necessary. The next byte (PARAM+6) is reserved for the status of the disk write on output.

On output, all parameters remain unchanged except for PARAM+6, which contains the status of the write. Status is 0 for a successful write, or nonzero if an error occurred during any portion of the write. If an error did not occur, the contents of the buffer has been written to the sector.





Algorithm

The disk drive number in L is first converted to the proper select configuration at WRD010. The select byte is then output to disk memory-mapped address 37E0H to select one of the disk drives.

The wait bit is then examined. If this bit is a zero, the loop at WRD015 counts HL through 65,536 counts to wait until the disk drive motor is up to speed before continuing.

The disk status is then examined (WRD020). If the disk is not busy, the track number is loaded into the disk controller track register (37EFH) and a seek command is given (37ECH) to cause the controller to "seek" the track for the operation. A series of time-wasting instructions is then done.

The code at WRD030 gets the disk status after completion of the seek and ANDs it with a "proper result" mask. If the status is normal, the write continues, otherwise an "abnormal" completion is done to WRD090.

The sector address from the parameter block is next output to the controller sector register (37EEH). Two time-wasting instructions are then done.

A write command is then issued to the disk controller command register (37ECH). Further time-wasting instructions are done.

The loop at WRD040 performs the actual write of the disk sector. A total of 256 separate writes is done, one for each byte. HL contains the disk address of 37ECH, DE contains a pointer to the buffer address, and BC contains the data register address of the disk controller. For each of the 256 reads, status is checked. If bit 0 is set, all 256 bytes have been written. If bit 1 of the status is set, the disk controller is still busy and a loop back to WRD040 is done. If bit 1 of the status is not set the next byte is read from memory, written to the disk, and the memory buffer pointer incremented.

At the automatic (by the controller) termination of the write, status is again read, and an AND of 7 is done to check for the proper completion bits. The status is stored back into the parameter block.

Sample Calling Sequence

```
NAME OF SUBROUTINE? WRDSEC
HL VALUE? 40000
PARAMETER BLOCK LOCATION? 40000
PARAMETER BLOCK VALUES?
+ 0  1  0      DRIVE 0
+ 1  1  20     TRACK 20
```

```
+ 2   1   5        SECTOR 5
+ 3   2   45000    BUFFER
+ 5   1   0        WAIT
+ 6   1   0
+ 7   0   0
MEMORY BLOCK 1 LOCATION?
MOVE SUBROUTINE TO? 38000
SUBROUTINE EXECUTED AT   38000
INPUT:                OUTPUT:
HL= 40000             HL= 40000
PARAM+ 0   0          PARAM+ 0   0
PARAM+ 1   20         PARAM+ 1   20
PARAM+ 2   5          PARAM+ 2   5      UNCHANGED
PARAM+ 3   200        PARAM+ 3   200
PARAM+ 4   175        PARAM+ 4   175
PARAM+ 5   0          PARAM+ 5   0
PARAM+ 6   0          PARAM+ 6   0      STATUS OK

NAME OF SUBROUTINE?
```

Notes

1. Always perform an RESTDS operation before initial disk I/O to initialize the disk controller.

Program Listing

```
7F00                         ORG     7F00H           ;0522
                    ;**********************************************************
                    ;* WRITE DISK SECTOR. WRITES BUFFER INTO SPECIFIED        *
                    ;* TRACK, SECTOR OF DISK.                                 *
                    ;*     INPUT: HL=> PARAMETER BLOCK                        *
                    ;*                 PARAM+0=DRIVE #, 0 - 3                 *
                    ;*                 PARAM+1=TRACK #, 0 - N                 *
                    ;*                 PARAM+2=SECTOR #, 0 - N                *
                    ;*                 PARAM+3,+4=BUFFER ADDRESS              *
                    ;*                 PARAM+5=0=WAIT AFTER SELECT, 1=NO WAIT *
                    ;*                 PARAM+6=STATUS, 0=OK, 1=BAD            *
                    ;*     OUTPUT:BUFFER WRITTEN TO TRACK, SECTOR             *
                    ;**********************************************************
                    ;
7F00 F5             WRDSEC   PUSH    AF              ;SAVE REGISTERS
7F01 C5                      PUSH    BC
7F02 D5                      PUSH    DE
7F03 E5                      PUSH    HL
7F04 DDE5                    PUSH    IX
7F06 CD7F0A                  CALL    0A7FH           ;***GET PB LOC'N***
7F09 E5                      PUSH    HL              ;TRANSFER TO IX
7F0A DDE1                    POP     IX
7F0C DD7E00                  LD      A,(IX+0)        ;GET DRIVE #
7F0F 3C                      INC     A               ;INCREMENT BY ONE
7F10 47                      LD      B,A             ;PUT IN B FOR CONVERT
7F11 3E80                    LD      A,80H           ;MASK
7F13 07             WRD010   RLCA                      ;ALIGN FOR SELECT
7F14 10FD                    DJNZ    WRD010            ;CONVERT TO ADDRESS
7F16 32E037                  LD      (37E0H),A       ;SELECT DRIVE
7F19 DD7E05                  LD      A,(IX+5)        ;GET WAIT/NO WAIT
7F1C B7                      OR      A               ;TEST
7F1D 2008                    JR      NZ,WRD020       ;GO IF NO WAIT
7F1F 210000                  LD      HL,0            ;WAIT COUNT
7F22 2B             WRD015   DEC     HL                ;DELAY LOOP 6
7F23 7D                      LD      A,L               ;TEST DONE 4
7F24 B4                      OR      H                 ;4
```

```
7F25 20FB      00460        JR      NZ,WRD015       ;LOOP UNTIL HL=0 7/12
7F27 3AEC37    00470 WRD020 LD      A,(37ECH)       ;GET STATUS
7F2A CB47      00480        BIT     0,A             ;TEST BUSY
7F2C 20F9      00490        JR      NZ,WRD020       ;LOOP IF BUSY
7F2E DD7E01    00500        LD      A,(IX+1)        ;GET TRACK NUMBER
7F31 32EF37    00510        LD      (37EFH),A       ;OUTPUT TRACK #
7F34 C5        00520        PUSH    BC              ;WASTE TIME
7F35 C1        00530        POP     BC
7F36 3E17      00540        LD      A,17H           ;SEEK COMMAND
7F38 32EC37    00550        LD      (37ECH),A       ;OUTPUT
7F3B C5        00560        PUSH    BC              ;WASTE TIME
7F3C C1        00570        POP     BC
7F3D C5        00580        PUSH    BC
7F3E C1        00590        POP     BC
7F3F 3AEC37    00600 WRD030 LD      A,(37ECH)       ;GET STATUS
7F42 CB47      00610        BIT     0,A             ;TEST BUSY
7F44 20F9      00620        JR      NZ,WRD030       ;LOOP IF BUSY
7F46 E698      00630        AND     98H             ;TEST FOR NORMAL COMPL
7F48 202C      00640        JR      NZ,WRD090       ;GO IF ABNORMAL
7F4A DD7E02    00650        LD      A,(IX+2)        ;GET SECTOR #
7F4D 32EE37    00660        LD      (37EEH),A       ;OUTPUT
7F50 C5        00670        PUSH    BC              ;WASTE TIME
7F51 C1        00680        POP     BC
7F52 21EC37    00690        LD      HL,37ECH        ;DISK ADDRESS
7F55 DD5E03    00700        LD      E,(IX+3)        ;PUT BUFFER ADDRESS IN DE
7F58 DD5604    00710        LD      D,(IX+4)
7F5B 3EAC      00720        LD      A,0ACH          ;WRITE COMMAND
7F5D 77        00730        LD      (HL),A          ;OUTPUT
7F5E C5        00740        PUSH    BC              ;WASTE TIME
7F5F C1        00750        POP     BC
7F60 C5        00760        PUSH    BC
7F61 C1        00770        POP     BC
7F62 01EF37    00780        LD      BC,37EFH        ;DATA REG ADDRESS
7F65 7E        00790 WRD040 LD      A,(HL)          ;GET STATUS
7F66 0F        00800        RRCA                    ;ALIGN
7F67 3008      00810        JR      NC,WRD050       ;GO IF DONE
7F69 0F        00820        RRCA                    ;ALIGN
7F6A 30F9      00830        JR      NC,WRD040       ;GO IF NOT DRQ
7F6C 1A        00840        LD      A,(DE)          ;GET BYTE
7F6D 02        00850        LD      (BC),A          ;OUTPUT TO DISK
7F6E 13        00860        INC     DE              ;INCREMENT MEMORY PNTR
7F6F 18F4      00870        JR      WRD040          ;LOOP TIL DONE
7F71 3AEC37    00880 WRD050 LD      A,(37ECH)       ;GET STATUS
7F74 E607      00890        AND     7               ;CHECK FOR PROPER STATUS
7F76 DD7706    00900 WRD090 LD      (IX+6),A        ;STORE STATUS
7F79 DDE1      00910        POP     IX              ;RESTORE REGISTERS
7F7B E1        00920        POP     HL
7F7C D1        00930        POP     DE
7F7D C1        00940        POP     BC
7F7E F1        00950        POP     AF
7F7F C9        00960        RET                     ;RETURN TO CALLING PROG
0000           00970        END
00000 TOTAL ERRORS
```

WRDSEC DECIMAL VALUES

```
245, 197, 213, 229, 221, 229, 205, 127, 10, 229,
221, 225, 221, 126, 0, 60, 71, 62, 128, 7,
16, 253, 50, 224, 55, 221, 126, 5, 183, 32,
8, 33, 0, 0, 43, 125, 180, 32, 251, 58,
236, 55, 203, 71, 32, 249, 221, 126, 1, 50,
239, 55, 197, 193, 62, 23, 50, 236, 55, 197,
193, 197, 193, 58, 236, 55, 203, 71, 32, 249,
230, 152, 32, 44, 221, 126, 2, 50, 238, 55,
```

197, 193, 33, 236, 55, 221, 94, 3, 221, 86,
4, 62, 172, 119, 197, 193, 197, 193, 1, 239,
55, 126, 15, 48, 8, 15, 48, 249, 26, 2,
19, 24, 244, 58, 236, 55, 230, 7, 221, 119,
6, 221, 225, 225, 209, 193, 241, 201

CHKSUM= 23

III
APPENDICES

APPENDIX I
Z-80 Instruction Set

The following is a brief explanation of the Z-80 instructions used in the TRS-80 subroutines. Refer to Zilog or Radio Shack documentation for more detailed descriptions.

ADC

This instruction adds one byte plus the current contents of the Carry flag to the contents of the A register when used in the format "ADD A,B"; the byte may be in another CPU register, an immediate value, or from memory. The instruction adds two bytes from a register pair plus the current contents of the Carry flag to the contents of HL, IX, or IY, when used in the format "ADD HL,DE." Flags are affected.

ADD

This instruction adds one byte to the contents of the A register when used in the format "ADD A,B"; the byte may be in another CPU register, an immediate value, or from memory. The instruction adds two bytes from a register pair, IX, or IY to the contents of HL, IX, or IY, when used in the format "ADD HL,DE." Flags are affected.

AND

This instruction logically ANDs one byte and the contents of the A register. The byte may be in a CPU register, an immediate value, or from memory. Typical format is "AND B," which ANDs the B and A registers. Flags are affected.

BIT

This instruction tests the bit of a CPU register or memory location. "BIT 7,B" tests bit 7 of the B register, while "BIT 0, (HL)" tests bit 0 of the memory location pointed to by the HL register pair. The state of the bit goes into the Carry flag.

CALL

This instruction calls a subroutine by pushing the return address into the stack. In the format "CALL 0212H" it is an unconditional call. In the format "CALL NZ,0212H" it is a conditional call. The conditions may be on the state of the Zero, Carry, Sign flag, or other flags. No flags affected.

CCF

This instruction complements the Carry flag; a set is changed to reset and vice versa.

CP

This instruction compares two bytes, one in the A register, and one from another CPU register or memory. The result does not replace the contents of A, but only sets the flags on the result of the compare. Typical format is "CP (HL)," which compares A with the contents of the memory location pointed to by the HL register pair. Flags are affected.

CPD

This instruction performs one step of an "end to beginning" block compare, using A as the comparison key, HL as the pointer, and BC as the number of bytes. Flags are affected.

CPDR

This instruction performs an "end to beginning" block compare, using A as the comparison key, HL as the pointer, and BC as the number of bytes. Flags are affected.

CPI

This instruction performs one step of a "beginning to end" block compare, using A as the comparison key, HL as the pointer, and BC as the number of bytes. Flags are affected.

CPIR

This instruction performs a "beginning to end" block compare, using A as the comparison key, HL as the pointer, and BC as the number of bytes. Flags are affected.

CPL

This instruction complements the contents of A; all ones are changed to zeroes, and all zeroes to ones. Most flags are unaffected.

DAA

This instruction adjusts the result in the A register so that it is a "decimal" or bcd result. Flags are affected.

DEC

This instruction decrements the contents of a CPU register by one, when used in the format "DEC E." When used in the format "DEC HL," it decrements the contents of a register pair by one. When used in the format "DEC (HL)" or "DEC (IX+5)" it decrements the contents of a memory location by one. Flags are affected only in the 8-bit case.

DI

This instruction disables interrupts.

DJNZ

This instruction decrements the contents of the B register and then jumps if the result is not zero. It is relocatable. Typical format is "DJNZ 9000H." Flags are unaffected.

EI

This instruction enables interrupts.

EX

This instruction swaps the contents of EX and HL when it is used in "EX DE,HL" or points to the "primed set" of the A register and flags when it is used in "EX AF,AF" or exchanges the first two bytes in the stack with HL, IX, or IY when used in "EX (SP),HL" format. Flags are unaffected.

EXX

This instruction switches to the primed set of BC, DE, and HL. Flags are unaffected.

IN

This is the input instruction. It inputs a value from an input/output device into the A register when in the form "IN A,(0FFH)." Flags are affected.

INC

This instruction increments the contents of a CPU register by one, when used in the format "INC E." When used in the format "INC HL," it increments the contents of a register pair by one. When used in the format "INC (HL)" or "INC (IX+5)" it increments the contents of a memory location by one. Flags are affected in 8-bit case only.

JP

This is the jump instruction. In the format "JP 9000H" or "JP (HL)," it is an unconditional jump. In the format "JP NZ,9000H," it is a conditional jump. The condition may be on the Zero flag (Z, NZ), Carry flag (C, NC), Sign flag (M, P), or other flags. Flags are unaffected.

JR

This is the jump "relative" instruction. It is identical in function to the "JP" instruction except that it is relocatable. Typical format is "JR 9000H" for an unconditional jump or "JR NZ,9000H" for a conditional jump. Flags are unaffected.

LD

This is the load instruction. It transfers data between CPU registers or between CPU registers and memory. When it is used to transfer data between two CPU registers, 8 bits will be transferred, and the format will be similar to "LD A,B" where B is the "source" and A is the destination. When it is used to transfer from a CPU register to memory, the format will be similar to "LD (3C00H),A" or "LD (HL),A"; the former transfers 8 bits from A to memory location 3C00H, the later transfers 8 bits from A to the memory location pointed to by HL. The format for 8 bit transfers from memory to a register will be reversed, as in "LD A,(3C00H)" and "LD A,(HL)."

LD can also be used to transfer 16 bits of data between a register pair and memory. The format will be similar to "LD HL,(3C00H)," which transfers the contents of location 3C00H and 3C01H to the L and H registers, respectively. To transfer data between memory and a register pair, the format is reversed as in "LD (3C00H),HL."

LD can also be used to transfer immediate data into a register or register pair, as in "LD A,45H," which loads A with 45H, or "LD HL,3C00H" which loads HL with the value 3C00H. Flags are unaffected.

LDD

This instruction performs one step of an "end to beginning" block move, using HL as the "source pointer," DE as the "destination pointer," and BC as the byte count. Flags are affected.

LDDR

This instuction performs one step of an "end to beginning" block move, using HL as the "source pointer," DE as the "destination pointer," and BC as the byte count. Flags are affected.

LDI

This instruction performs one step of a "beginning to end" block move, using HL as the "source pointer," DE as the "destination pointer," and BC as the byte count. Flags are affected.

LDIR

This instruction performs a "beginning to end" block move, using HL as the "source pointer," DE as the "destination pointer," and BC as the byte count. Flags are affected.

NEG

This instruction takes the two's complement of the A register. It "negates" the contents of A. Flags are affected.

NOP

This instruction is a "no operation" performing no function. Flags are unaffected.

OR

This instruction logically ORs one byte and the contents of the A register. The byte may be in a CPU register, an immediate value, or from memory. Typical format is "OR B," which ORs the B and A registers. Flags are affected.

OUT

This is the output instruction. It outputs a byte from the A register to an input/output device when in the form "OUT (0FFH),A." Flags are unaffected.

POP

This instruction POPs a two-byte value from the stack and puts it into a register pair. "POP DE" loads the D and E registers with the next two bytes from the stack and adjusts the SP register by two. Flags are unaffected unless AF POPped.

PUSH

This instruction pushes a register pair, IX, or IY onto the stack. "PUSH BC" pushes the contents of B and C onto the stack and adjusts the SP register by two. Flags are unaffected.

RES

This instruction resets a bit in a CPU register or memory location. "RES 5,A" resets bit 5 of the A register to 0, while "RES 2,(HL)" resets bit 2 of the memory location pointed to by the HL register pair. Flags are unaffected.

RET

This instruction returns from a subroutine by popping the return address from the stack. If the format is "RET," it is an unconditional return; if the format is "RET NZ," the return is conditional upon the Zero, Carry, Sign, or other flags. Flags are unaffected.

RL

This instruction rotates the contents of a CPU register and carry (nine bits) left one bit position. Typical format is "RL D" which rotates the D register and carry. Flags are affected.

RLA

This instruction rotates the A register and carry (nine bits) one bit position left. Flags are affected.

RLC

This instruction rotates the contents of a CPU register one bit position left. Typical format is "RLC E," which rotates the E register. Flags are affected.

RLCA

This instruction rotates the A register one bit position left. Flags are affected.

RLD

This instruction rotates the memory location pointed to by HL and the least significant four bits of the A register four bits left. It is a "bcd shift." Flags are affected.

RR

This instruction rotates the contents of a CPU register and carry (nine bits) one bit position right. Typical format is "RR B" which rotates the B register and carry. Flags are affected.

RRA

This instruction rotates the A register and carry (nine bits) one bit position right. Flags are affected.

RRC

This instruction rotates the contents of a CPU register one bit position right. Typical format is "RRC H," which rotates the H register. Flags are affected.

RRCA

This instruction rotates the A register one bit position right. Flags are affected.

RRD

This instruction rotates the memory location pointed to by HL and the least significant four bits of the A register four bits right. It is a "bcd shift." Flags are affected.

SBC

This instruction subtracts one byte minus the current contents of the Carry flag from the contents of the A register when used in the format "SBC A,B"; the byte may be in another CPU register, an immediate value, or from memory. The instruction subtracts two bytes from a register pair minus the current contents of the Carry flag from the contents of HL, IX, or IY, when used in the format "SBC HL,DE." Flags are affected.

SCF

This instruction sets the Carry flag.

SET

This instruction sets a bit in a CPU register or memory location. "SET 5,C" sets bit 5 of the C register, while "SET 0,(HL)" sets bit 0 of the memory location pointed to by the HL register pair. Flags are unaffected.

SLA

This instruction logically shifts a CPU register one bit position left. Typical format is "SLA H," which shifts the H register. Flags are affected.

SRA

This instruction arithmetically shifts a CPU register one bit position right. Typical format is "SRA A," which shifts the A register. Flags are affected.

SRL

This instruction logically shifts a CPU register one bit position right. Typical format is "SRL L," which shifts the L register. Flags are affected.

SUB

This instruction subtracts one byte from the contents of the A register when used in the format "SUB A,B"; the byte may be in another CPU register, an immediate value, or from memory. The instruction subtracts two bytes from a register pair, IX, or IY from the contents of HL, IX, or IY, when used in the format "SUB HL,DE." Flags are affected.

XOR

This instruction logically exclusive ORs one byte and the contents of the A register. The byte may be in a CPU register, an immediate value, or from memory. Typical format is "XOR B," which XORs the B and A registers. Flags are affected.

APPENDIX II
Decimal/Hexadecimal Conversion

0	00	64	40	128	80	192	C0
1	01	65	41	129	81	193	C1
2	02	66	42	130	82	194	C2
3	03	67	43	131	83	195	C3
4	04	68	44	132	84	196	C4
5	05	69	45	133	85	197	C5
6	06	70	46	134	86	198	C6
7	07	71	47	135	87	199	C7
8	08	72	48	136	88	200	C8
9	09	73	49	137	89	201	C9
10	0A	74	4A	138	8A	202	CA
11	0B	75	4B	139	8B	203	CB
12	0C	76	4C	140	8C	204	CC
13	0D	77	4D	141	8D	205	CD
14	0E	78	4E	142	8E	206	CE
15	0F	79	4F	143	8F	207	CF
16	10	80	50	144	90	208	D0
17	11	81	51	145	91	209	D1
18	12	82	52	146	92	210	D2
19	13	83	53	147	93	211	D3
20	14	84	54	148	94	212	D4
21	15	85	55	149	95	213	D5
22	16	86	56	150	96	214	D6
23	17	87	57	151	97	215	D7
24	18	88	58	152	98	216	D8
25	19	89	59	153	99	217	D9
26	1A	90	5A	154	9A	218	DA
27	1B	91	5B	155	9B	219	DB
28	1C	92	5C	156	9C	220	DC
29	1D	93	5D	157	9D	221	DD
30	1E	94	5E	158	9E	222	DE
31	1F	95	5F	159	9F	223	DF
32	20	96	60	160	A0	224	E0
33	21	97	61	161	A1	225	E1
34	22	98	62	162	A2	226	E2
35	23	99	63	163	A3	227	E3
36	24	100	64	164	A4	228	E4
37	25	101	65	165	A5	229	E5
38	26	102	66	166	A6	230	E6
39	27	103	67	167	A7	231	E7
40	28	104	68	168	A8	232	E8
41	29	105	69	169	A9	233	E9
42	2A	106	6A	170	AA	234	EA
43	2B	107	6B	171	AB	235	EB
44	2C	108	6C	172	AC	236	EC
45	2D	109	6D	173	AD	237	ED
46	2E	110	6E	174	AE	238	EE
47	2F	111	6F	175	AF	239	EF
48	30	112	70	176	B0	240	F0
49	31	113	71	177	B1	241	F1
50	32	114	72	178	B2	242	F2
51	33	115	73	179	B3	243	F3
52	34	116	74	180	B4	244	F4
53	35	117	75	181	B5	245	F5
54	36	118	76	182	B6	246	F6
55	37	119	77	183	B7	247	F7
56	38	120	78	184	B8	248	F8
57	39	121	79	185	B9	249	F9
58	3A	122	7A	186	BA	250	FA
59	3B	123	7B	187	BB	251	FB
60	3C	124	7C	188	BC	252	FC
61	3D	125	7D	189	BD	253	FD
62	3E	126	7E	190	BE	254	FE
63	3F	127	7F	191	BF	255	FF

PERSONAL COMPUTERS

William Barden, Jr.

TRS 80 Assembly Language Subroutines

Here is a hands-on approach to programming that explains how any TRS-80 computer user can increase productivity and reduce the tediousness of programming by using assembly-language subroutines.

TRS-80 ASSEMBLY LANGUAGE SUBROUTINES uses the speed and compactness of assembly-language programming and gives you fully debugged, ready-to-run subroutines, including:
• a subroutine that converts binary numbers in memory to decimal characters • a subroutine that generates high-speed clearing of a screen block • a subroutine that outputs music through the cassette port in seven octaves • a subroutine that generates pseudo-random numbers for simulation or modeling • a subroutine that generates high-speed string searches

Each of the 65 fully documented subroutines includes:
• a complete description of what the subroutine does • the input/output parameters required to use the subroutine • the algorithm for the subroutine • a sample calling sequence • notes on special uses or features • a decimal listing • a "check" on the validity of the data.

PRENTICE-HALL, Inc., Englewood Cliffs, New Jersey 07632

TRS-80

ISBN 0-13-931188-2