Z80 INSTRUCTION HANDBOOK

NAT WADSWORTH

Your complete reference to the powerful Z80 instruction set.

HAYDEN

Z80 INSTRUCTION HANDBOOK

NAT WADSWORTH

HAYDEN BOOK COMPANY, INC.
Hasbrouck Heights, New Jersey

4 5 6 7 8 9 PRINTING

83 84 85 86 87 88 89 90 YEAR

Preface

Some years ago when I described the instructio set for what has now often been classifieu as the first true microcomputer — the 8008 by Intel Corporation — I wrote that its irst [illegible]ion set of about 170 instruct'ons was "quite comprehensive."

Well 't was — in those days it was revolutionary! But time does not stand still. And now, just a few short years later, I find myself describing the instruction set for a CPU-on-a-chip that has "quite a comprehensive" instruction set that numbers in the vicinity of 700 (yes, *seven hundred*) instructions. And I know as I write thi- that in a few more years, 700 instructions won't be considered anywhere near "comprehensive" for the soon to be available 16-bit microprocessors.

If one stops to soberly consider the present state of affairs in this field, it soon becomes apparent that the profusion of microprocessor products is rapidly outstripping our ability to utilize them at anywhere near their full potential. It seems that long before the average programmer has become comfortably familiar with the instruction capabilities of one CPU, there is another device on the scene beckoning with higher speeds and greater capabilities. I sometimes seriously question the merits of this process. However, the forces at work in our society seem to inexorably move us in that direction. Since that is the case it seems that the user of these marvelous devices must find ways of rapidly learning the instructional capabilities of a device. The motivation for this may be professional, such as when one must design programs for a particular machine. Or academic, such as when one merely wants to know a general capability.

The purpose of this publication is to explain, in easily understood terms, the capabilities of the instruction set of the Z80 CPU (produced by Zilog Corporation and other second source manufacturers). This handbook serves as a practical reference to the industry standard mnemonics, machine code, and usage for each type of instruction provided in the Z80 CPU. It is meant to serve as a practical guide for the novice, intermediate, or professional programmer who has a requirement to work at the machine or assembler language level with a Z80 based microprocessor.

To gain an overview of the Z80's instruction set I would suggest that the user first lightly skim the material as a text. To use the book as a programming and program assembly aid, please take note of the alphabetically organized index provided in Appendix A.

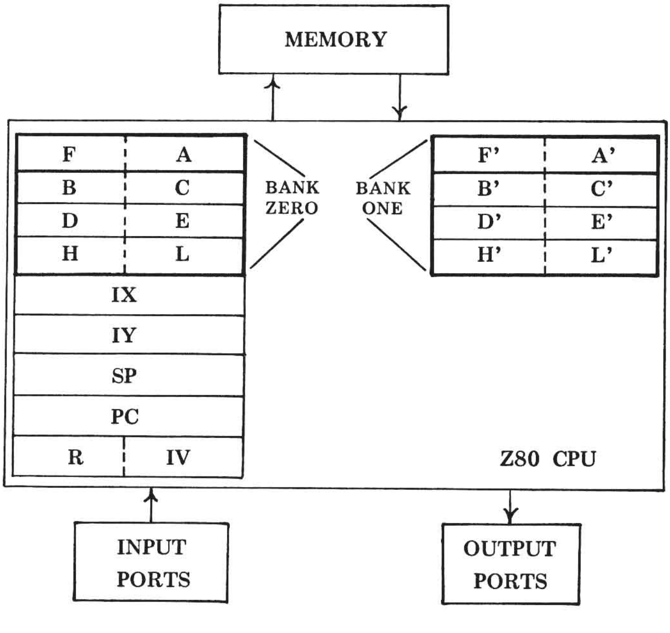
THE Z80 CPU INSTRUCTION SET

The Z80 CPU has a comprehensive instruction set that consists of some 245 basic instructions. When the possible permutations are considered, the total number of valid instruction codes numbers approximately seven hundred

The Z80 instruction set includes as a subset the entire instruction set of its precursor CPU — the 8080. Indeed, the subset of 8080 instructions (with the exception of a few directives involving the Parity Fla) is identical at the machine code level. Unfortunately, the industry standard mnemonics promulgated by the originator of the Z80; a corporation based in California using the trade name Zilog, Incorporated; are not at all similar to those in use by 8080 users. Hence, while it is possible in most cases to load programs in machine code previously developed for 8080 systems and have them execute without modification on a Z80 system, at the assembly language level the 8080 programmer must learn an entirely new set of mnemonics when working with the Z80 instruction set. To avoid confusing those who may not be familiar with 8080 mnemonics this book will utilize Z80 mnemonics as promulgated by the CPU's developer. Users who have 8080 assembly language experience and who may desire a mnemonic cross reference between Z80 and 8080 instructions might be advised to see Adam Osborne's *An Introduction to Microcomputers, Volume II.* Chapter 7 of the June, 1977 revised edition contains such a cross-referenced table. Additionally that publication contains extensive hardware information on the Z80 CPU. That is a subject that will not be covered in this publication, but which may be of interest to many readers. Also, that publication provides many comparisons of the Z80 to the 8080/8085 which may be highly beneficial to 8080 users who are changing over to the Z80.

The Z80 instruction set enables the programmer to direct the operation of the CPU to transfer and perform mathematical and logical operations on the data stored in the various memory and register elements that make up the computer. These software related elements are the memory, the program counter (PC), the stack pointer (SP), two index registers (designated IX and IY), an interrupt vector register (IV), a refresh register (R), two banks of independent and interchangeable "working registers" consisting of a flag status register (F), an accumulator (A), and six general registers (B, C, D, E, H, L); and the input and output ports. Within each "bank" of working registers the accumulator and its associated flag status register can independently be activated or deactivated. Thus there are four possible arrangements of the working registers, of which one may be selected at any given time.

1.) All the registers in Bank Zero.
2.) All the registers in Bank One.
3.) The accumulator and flags of Bank Zero operating with registers B — L of Bank One.
4.) The accumulator and flags of Bank One operating with registers B — L of Bank Zero.



The memory is the program storage element for the CPU. When a program is to be executed, its instructions must first be stored in the proper locations in memory. The memory is also used to store data which the program will require during its execution. For the Z80, up to 65,536 eight-bit bytes of memory can be directly accessed by the CPU.

The program counter is a sixteen bit register used to control the flow of a program from one instruction to the next. When a program is started, the program counter is set to the address in memory of the first instruction to be executed. Unless directed otherwise by the instruction just executed, the program counter will automatically increment to the address of the next instruction in sequence and execute it. When an instruction that directs the program to an address other than the next sequential location is executed, the new address is placed in the program counter and program execution is continued with the instruction at the new address.

The next group of CPU registers to be described have been assigned letters so that they may be referred to symbolically. The accumulator register is denoted by the letter A. Six general registers are referred to as the B and C, D and E, and H and L registers. These six registers may serve as individual eight-bit storage registers capable of holding data and performing limited manipulations such as incrementing or decrementing their contents. Alternatively, they may be coupled together as indicated by their grouping to serve as 16-bit registers capable of performing limited operations such as pointing to locations in memory for classes of memory reference instructions, or serving as double-precision counters, etc. When coupled to form 16-bit registers, those designated B, D and H form the most significant half with the remaining C, E and L registers forming the corresponding least significant halves of the two-byte values.

The accumulator is an eight-bit register. It is the workhorse of the Z80 in that most major mathematical and logical functions can only be performed with the accumulator.

The register referred to by the letter F serves as a flag status register. It will be described in further detail snortly. In the block diagram showing the CPU registers it is shown residing next to the accumulator register. This was done only because the contents of the F register may be transferred using selected "stack" directives in parallel with the contents of the accumulator as the consequence of a single directive. This apparently was a design convenience because all Z80 stack operations transfer two bytes of data at a time. It was thus operationally convenient to group the one-byte flag status register with the single-byte accumulator when making such stack transfers. Other than this consideration the flag and accumulator registers should be considered as separate entities (though the condition of various bits within the flag status register are influenced by the contents of the accumulator among other registers as will become apparent soon).

As indicated previously and illustrated in the block diagram of the available CPU registers, there are two sets or "banks" of the registers. Each bank is further subdivided in regards to the accumulator versus the remaining "working registers." While the registers in only one group (accumulator/working registers in Bank 0/Bank 1) can be activated at any given time, the information in the remaining "inactive" registers will be retained as long as power is applied to the CPU. Thus, registers in the inactive mode may serve as storage locations for data that can quickly be brought "on-line." Just a single instruction is required to designate which bank of general registers (or accumulator) is to be in the active mode at any instant.

The two index registers symbolically referred to as IX and IY are provided to allow indexed addressing. A 16-bit value in either register may be added to a one-byte offset value (included as part of all instructions when using the indexed addressing mode) to form an effective address. The effective address so formed points to the location in memory that will be acted upon by the instruction. The availability of two independent 16-bit index registers is a great boon when developing programs that must shift data between various sections in memory. One register can point to a data source area in memory while the other services a data destination region, etc.

The memory pointer register used to designate a section of memory in which to store the return addresses of subroutines and for temporary storage of other data is called the "stack pointer." The stack pointer is a 16-bit register that may point to any location ın memory that is to be used as a program "stack." A "stack" in this context refers to a section of memory in which subroutine return addresses are automatically stored by the CPU or in which other data may be saved in a push-down/pop-up manner by transferring the contents of various CPU registers to/from memory locations designated by the contents of the stack pointer.

A major function of almost every computer program is to receive or transmit data between the computer and one or more peripheral devices. In the Z80 this is accomplished through the use of input and output ports. There can be up to 256 input and 256 separate output ports in a Z80 system. Each input port and output port provides eight parallel data lines for communicating with external devices.

In addition to the traditional microcomputer capability which provides for transferring information between an input/output (I/O) port and an accumulator register, the Z80 has instructions which allow such transfers to take place directly between I/O ports and instruction-designated CPU working registers. Or, such transfers can be directed between memory locations and I/O ports. What's more, block I/O transfers may be executed so that multiple-byte transfers can take place between memory locations and an I/O device. Such transfers can be initiated with just a single Z80 instruction.

CPU FLAGS

The Z80 CPU has a number of special flip-flops which are referred to as "flags." These flip-flops are set or cleared as the result of specific operations. They are important because they may be tested by a group of "conditional" instructions provided for the Z80. The ultimate action such a conditional instruction takes will be dictated by the status of a corresponding flag at the time the instruction is actually executed. These instructions endow the computer with the ability to make "decisions." The Z80 CPU is provided with six basic "flags." The operation of these flags and their symbolic names are described next.

The carry (C) flag refers to the status of a one-bit cell representing the overflow or underflow from the accumulator. It can also be set to a desired condition by certain types of instructions. This is important to remember when developing programs. Since many types of instructions affect the status of the carry bit, it may be difficult to predict the status of the carry bit after a series of instructions has been executed. However, when a program reaches a point where the status of the carry bit must be relied upon, an appropriate directive may be used to set the flag to a specific condition.

The half (H) carry flag refers to the "half carry" bit status. The half carry is a one unit register (flip-flop) that is used to indicate when an overflow or underflow from bit B3 occurs. (This publication will reference bit B0 as the least significant bit in a register. Thus bit B3 designates the *fourth* bit in a register — B0 being the *first.*) The H flag is affected by addition, subtraction, increment, decrement, and compare instructions. It may be altered by other types of directives as pointed out elsewhere in this manual. Unlike most of the other status flags, it cannot be tested by other instructions. Its only function is to internally serve the CPU when performing selected types of decimal arithmetic operations such as a decimal adjust of the accumulator after a mathematical operation. Such an operation can convert the binary contents of the register to two binary-coded-decimal (BCD) digits. Details of such a conversion will be outlined later in this publication.

The zero (Z) flag refers to a flip-flop that can indicate whether the value of the accumulator or other associated register is exactly equal to zero. It is set to a logic one condition if the register is zero. It is cleared to a logic zero if the related register is non-zero. Note the converse relationship. It is sometimes confusing to beginning machine language programmers! This flag is also used during the execution of bit-test instructions to indicate the complemented state of the bit being tested. That is, if the tested bit is zero, the flag will be one and vice-versa. Once again — take note of the converse relationship! The Z flag is also affected by selected Z80 I/O operations as will be detailed elsewhere.

The sign (S) flag refers to a flip-flop that indicates whether the value in the accumulator or other associated register is a positive or negative value based on the two's complement convention. Essentially, this flag monitors the most significant bit in the accumulator or other associated register and mimics its condition.

The next flag to be discussed serves as a combination parity and arithmetic overflow (P/V) indicator. When arithmetic instructions are being performed this flag functions as an overflow detector. The Z80 CPU utilizes the two's complement arithmetic convention whereby the most significant bit in a register indicates the sign (zero for positive, one for negative) of the value held by the remaining bits in the register. If adding two numbers having like signs results in the register changing sign, the P/V flag will be set. If not, it will be cleared. Conversely, if subtracting two numbers having different signs results in the minuend (number from which a value is being subtracted) register (assumed to hold the result of the subtraction operation) changing sign then the P/V flag will be set. Otherwise it is cleared to the logic zero condition. The P/V flag can also serve as a parity indicator during the execution of such operations as register rotates and Boolean logic functions when it is not being used to indicate arithmetic overflow. In such instances it is set to indicate that the accumulator or other associated register contains a value that has even parity. Parity is useful for a number of reasons and is usually used in conjunction with testing for error conditions on words of data, particularly when inputting data from external sources. Even parity occurs when the number of bits that are set to the logic one state (out of eight possible) is an even count (0, 2, 4, 6 or 8) regardless of their positions within the register. The P/V flag is also affected by the execution of block transfer and search directives, and some other special types of instructions. These cases will be discussed as appropriate later.

The final flag to be mentioned here is called the add/subtract (N) flag. It is similar to the H flag mentioned earlier in that it is not accessible to a programmer. It is used internally by the CPU to enable it to distinguish between addition (which clears the flag to a logic zero state) and subtraction (which sets the flag to a logic one state) instructions and is required when such mathematical operations are followed by a decimal adjust directive. The existence of this flag and its status is purely academic as far as the programmer is concerned.

It is important to note that the Z, S and P/V flags (as well as the previously mentioned C flag) can all be set to known states by appropriate instructions. It is also important to note that some instructions do not result in the flags being set so that if the programmer desires to have a routine make decisions based on the status of flags, the programmer must make sure that the proper instructions, or sequence of instructions, is utilized. It is important to note in particular that "load register" directives do not themselves affect the flags. Since it is often desirable to obtain a data word (i.e., load it into an accumulator) and test its status for such information as whether or not the value is zero, a negative number, and so forth; the programmer must remember to follow a load instruction in such a situation by a logical instruction (such as the AND A — "logical AND the accumulator with itself") in order to set up the Z, S and P/V flags (the C flag would not be affected by such a directive) before using a "conditional instruction."

NOTATION

The descriptions of the various types of instructions available on a Z80 CPU that are provided herein will include the mnemonics and machine language codes for the instructions. The machine codes will be given in two forms: three digit octal codes and two digit hexadecimal notation. It may be noted that the mnemonics used throughout this book are those originally promulgated by Zilog Corporation, the developers of the Z80 CPU. Despite the fact that many 8080 users, noting that the Z80 is an enhancement of that device, may be disappointed that Zilog Corporation did not expand the widely accepted industry standard 8080 mnemonics, this book will not attempt to alter the path that history has shown is generally followed. The originator of the device almost always sets a de facto standard for the mnemonic nomenclature that becomes universally accepted. There are already second sources for the Z80. These manufacturers are adopting the same general mnemonic symbols. As an aid to users of this publication, Appendix A lists the mnemonics used in this publication in alphabetical order. It may be used to quickly locate the detailed description of an instruction in this section. If the programmer is not already aware of it, the use of mnemonics facilitates working with an "assembler" program when it is desired to develop relatively large and complex programs. Thus, the programmer is urged to concentrate on learning the mnemonics for the instructions and not waste time memorizing the machine codes. After a program has been written using the mnemonic codes, the programmer can always use a lookup table to convert to the machine code if an assembler program is not available. It is a lot easier technique (and subject to less error) than trying to memorize the some seven hundred digital combinations that make up the machine code instruction set!

The machine code for the instructions is presented in both three digit octal and two digit hexadecimal notation. Both formats are provided for those readers that may be familiar with only one or the other. As may be observed during the presentation of the instructions, the three digit octal format allows easier recognition of many of the types of instructions — particularly when it comes to specifying operand registers, than does the two digit hexadecimal representation. It may also be easier for a novice to deal with numbers that only range from zero to seven instead of having to deal with the ramifications of tacking letters of the alphabet onto the numbering system as is done with hexadecimal notation.

The programmer must be aware that many Z80 instructions require more than one byte in order to be fully specified. Some of the "immediate" type commands, classes of instructions using indexed and relative addressing modes, and input and output instructions require two consecutive bytes. There are also many directives such as those that identify an operand as being in a specific memory location that require three or four bytes. The number of bytes required for each type of instruction will be indicated in the descriptions that follow.

LOAD AND DATA TRANSFER INSTRUCTIONS

The first group of instructions to be presented consists of those used to load data from a single CPU register to another CPU register, or from a single CPU register to a location in memory, or vice-versa. This group of instructions requires just one byte. It is important to note that none of the instructions in this group affect the CPU flags.

LOAD DATA FROM ONE CPU REGISTER TO ANOTHER CPU REGISTER

MNEMONIC	OCTAL	HEXADECIMAL
LD A,A	177	7F
LD B,A	107	47
.	.	.
.	.	.
LD A,B	170	78

The load register group of instructions allows the programmer to move the contents of one CPU register into another CPU register. The contents of the originating (from) register are not changed. The contents of the destination (to) register become the same as the originating register. Any CPU register may be loaded into any CPU register. Note that, for instance, loading register A into register A is essentially a "no operation" directive. When using mnemonics, the load symbol is the mnemonic LD followed by the destination register and then the source register. The mnemonic LD B,A means that the contents of register A (the accumulator) is to be loaded into register B. The mnemonic LD A,B states that register B is to have its contents loaded into register A. It may be observed that this basic instruction has many variations. The octal representation of the machine language coding for this instruction has the same format as the mnemonic code except that the letters used to represent the registers in the mnemonic form are replaced by octal digits representing binary patterns that the CPU can recognize in the machine language form. Using octal code, the seven CPU registers are coded as follows:

REG A	=	7
REG B	=	0
REG C	=	1
REG D	=	2
REG E	=	3
REG H	=	4
REG L	=	5
MEM	=	6

The last entry in the above list illustrates that memory reference instructions, to be presented shortly, code to the binary pattern represented by the octal number six.

By observing the entire set of machine codes representing load directives one may discern for this class of instructions, that when the most significant octal digit in the machine code is a one, it signifies that the computer is to perform a load operation. The next two octal digits completing the code specify the destination and source registers in order. Thus, in machine code, the instruction to load register B with the contents of register A has the octal value 107. The use of hexadecimal code does not lend itself as readily to this method of direct machine language coding because the binary group of three bits represented by the middle octal digit in the above discussion is split between the two 4-bit binary groups that map to hexadecimal notation.

It is important to note that the load instructions above do not affect any of the CPU flags.

LOAD DATA FROM ANY CPU REGISTER
TO A LOCATION IN MEMORY

LD (HL),A	167	77
LD (HL),B	160	70
LD (HL),C	161	71
LD (HL),D	162	72
LD (HL),E	163	73
LD (HL),H	164	74
LD (HL),L	165	75

This type of instruction is similar to the previous group of instructions except that now the contents of a CPU register are loaded into a specific memory location. The memory location that will receive the contents of the designated CPU register is that whose address is contained in the CPU register pair H and L at the time the instruction is encountered. The H register specifies the high portion of the memory address desired while the L register specifies the low part of that address. Note that there are seven different instructions in this group as any CPU register may have its contents loaded into any location in memory. Instructions in this group do not alter the CPU flags.

LOAD DATA FROM A MEMORY LOCATION TO ANY CPU REGISTER

LD A,(HL)	**176**	**7E**
LD B,(HL)	**106**	**46**
LD C,(HL)	**116**	**4E**
LD D,(HL)	**126**	**56**
LD E,(HL)	**136**	**5E**
LD H,(HL)	**146**	**66**
LD L,(HL)	**156**	**6E**

This group of instructions may be considered as the complement of the immediately preceeding group. Now, the contents of a byte in memory whose address is specified by the H (for the high portion of the address) and L (low part) registers will be loaded into the CPU register specified by the instruction. Once again, this group of instructions has no affect on the status of the flags.

LOAD IMMEDIATE DATA INTO A CPU REGISTER

LD A,ddd	**076 ddd**	**3E dd**
LD B,ddd	**006 ddd**	**06 dd**
LD C,ddd	**016 ddd**	**0E dd**
LD D,ddd	**026 ddd**	**16 dd**
LD E,ddd	**036 ddd**	**1E dd**
LD H,ddd	**046 ddd**	**26 dd**
LD L,ddd	**056 ddd**	**2E dd**

This type of "immediate data" instruction requires two consecutive bytes in order to be completely specified. The first byte contains the operation code for the instruction. The second byte — the byte "immediately following" the operation code — must contain the data upon which action is taken. (The term "ddd" as used in the above group of instructions will be used in this text to denote a general byte of data.) Thus, a load immediate instruction in this context will cause the CPU to load the contents of the byte immediately following the instruction code into the designated CPU register. The load immediate directive LD A,1 would result in the value 00000001 (binary) being placed in the accumulator when the instruction was executed.

It is important to remember that all immediate type directives must be followed by a data byte. An instruction such as LD A alone (instead of LD A,ddd) would result in improper operation. This is because the computer would assume the next byte contained data. If the programmer had mistakenly left out the data and in its place had another instruction, the computer would not realize the operator's mistake. Hence, the program would be fouled up.

Note too, that the load immediate class of instructions does not affect the status of the CPU flags.

LOAD IMMEDIATE DATA INTO A MEMORY LOCATION

LD (HL),ddd	**066 ddd**	**36 dd**

This instruction is essentially the same as the load immediate group that transfers the data into a CPU register except that now the contents of the H and L registers are used as a pointer to an address in memory. The contents of the data byte will be placed in the specified memory location. This instruction does not affect the status of the CPU flags.

LOAD INSTRUCTIONS THAT USE INDEX REGISTERS

The next six types of load directives to be presented involve the IX and IY index registers. There are several points worthy of note regarding these instructions. First, the directives themselves require two bytes just to specify the operation code, A third byte is required for the offset value that all indexed instructions must provide. For immediate type instructions, a fourth byte will be used to provide "immediate data."