MOSTEK Z80 PROGRAMMING MANUAL

MOSTEK®

Z80 MICROCOMPUTER SOFTWARE

Programming Guide

4.50

Z80 PROGRAMMING MANUAL

PROGRAMMING MANUAL

FOR

Z80 MICROCOMPUTER

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

ERRATA

The following changes apply to the Z80 PROGRAMMING MANUAL, MK78515, ISSUE DATE MAY 1977.

1. For the RETI instruction (page 2-198), the statement which reads, "This instruction also resets the IFF1 and IFF2 flip flops." is incorrect. It should read, "This instruction has no effect on IFF1 and IFF2."

2. For block transfer/search instructions LDIR - LDDR - CPIR - and CPDR (pages 2-168, 2-164, 2-63, and 2-59 respectively) and for block input/output instructions INIR - INDR - OTIR - and OTDR (pages 2-104, 2-100, 2-177, and 2-175 respectively), the statement which reads, "Also, interrupts will be recognized after each data transfer/comparison". should read, "Interrupts will be recognized and two refresh cycles will be executed after each data transfer/comparison."

3. For subtract instructions SUB S - SBC A,s - SBC HL,ss - NEG - DEC m - CPDR - CPD - CP s - CPI - and CPIR (pages 2-254, 2-234, 2-235, 2-170, 2-71, 2-60, 2-57, 2-56, 2-61, and 2-64 respectively), the statement under Condition Bits Affected: "H: Set if no borrow from Bit _; reset otherwise" is incorrect. It should read, "H: set if there is a borrow from Bit _; reset otherwise".

   Also, for instructions SUB S - SBC A,s - SBC HL,ss - and CP s , the statement which reads, C: Set if no borrow; reset otherwise " is incorrect. It should read, "C: Set if there is a borrow, reset otherwise". For instructions CPDR - CPD - CPIR - and CPI , C is unaffected.

On page 2-8, paragraph 2-34, the statement which reads "For ADD instructions that generate a carry and for SUBTRACT instructions that generate no borrow, the Carry Flag will be set. The Carry Flag is reset by an ADD that does not generate a carry and a SUBTRACT that generates a borrow. " is incorrect. It should read, "For ADD instructions that generate a Carry and for SUBTRACT instructions that generate a borrow, the Carry Flag will be set. The Carry Flag is reset by an ADD that does not generate a carry and a SUBTRACT that does not generate a borrow.".

Within the table of paragraph 2-47 on page 2-10, the statements in the SUBTRACT column are incorrect and should read as follows: for H=1, "There is a borrow from bit 4", and for H=0, "There is no borrow from bit 4".

PREFACE

This manual is designed to help the user program the Z80 microcomputer in assembly language. It also serves as a standard for the Z80 assembly language.

It is assumed that the user has a background in logic and some experience with programming.

The manual consists mainly of a brief general description of the Z80 CPU architecture from the programmer's point of view and a detailed description of the Z80 instruction set. The description of the instruction set includes a description of the set's main features, specific information about assembly language syntax, and detailed descriptions of each of the Z80 opcodes. The manual also contains several appendices. Appendix A is an alphabetical list of the Z80 opcodes. Appendix B provides details of the Mostek assembler standard pseudo-ops. Appendix C describes the Mostek standard Z80 object code format. Appendix D provides binary, hexadecimal, and ASCII reference tables.

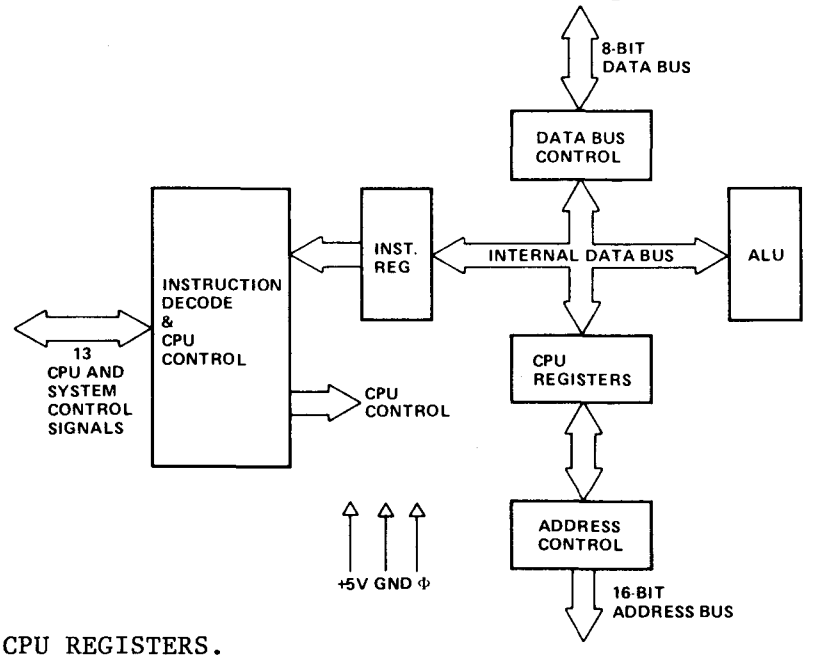
SECTION I

Z80 CPU ARCHITECTURE

1-1. INTRODUCTION.

1-2. A block diagram of the internal architecture of the Z80 CPU is shown in Figure 1-1. The diagram shows all of the major elements in the CPU and it should be referred to throughout the following description.

Figure 1-1. Z80 CPU Block Diagram



1-3. CPU REGISTERS.

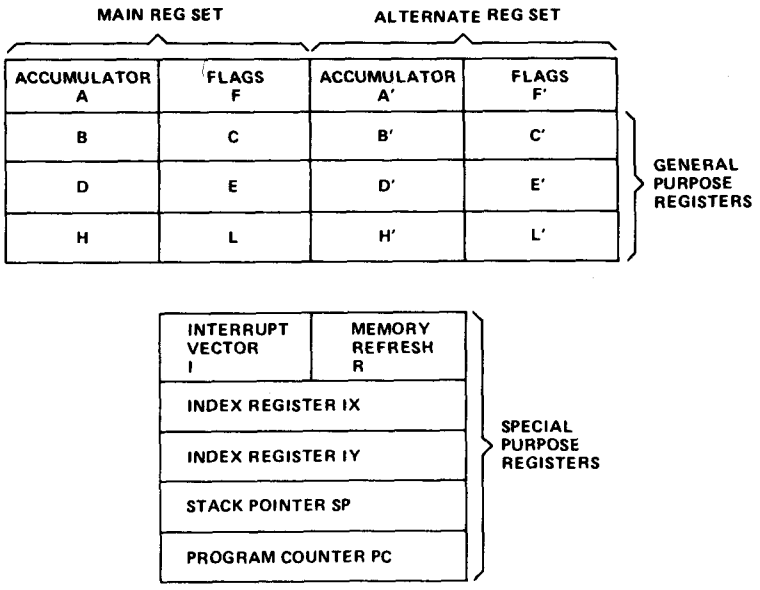
1-4. The Z80 CPU contains 208 bits of R/W memory that are accessible to the programmer. Figure 1-2 illustrates how this memory is configured into eighteen 8-bit registers and four 16-bit registers. All Z80 registers are implemented using static RAM. The registers include a set of special purpose registers, two sets of accumulator and flag registers, and two sets of six general purpose registers which may be used individually as 8-bit registers or in pairs as 16-bit registers.

1-5. SPECIAL PURPOSE REGISTERS.

1-6. Program Counter (PC). The program counter holds the 16-bit address of the current instruction being fetched from memory. The PC is automatically incremented after its contents have been transferred to the address lines. When a program jump occurs, the new value is automatically placed in the PC, overriding the incrementer.

1-7. Stack Pointer (SP). The stack pointer holds the 16-bit address of the current top of a stack located anywhere in external system RAM memory. The external stack memory is organized as a last-in first-out (LIFO) file. Data can be pushed onto the stack from specific CPU registers or popped off of the stack into specific CPU registers through the execution of PUSH and POP instructions. The data popped from the stack is always the last data pushed onto it. The stack allows simple implementation of multiple level interrupts, unlimited subroutine nesting and simplification of many types of data manipulation.

Figure 1-2. Z80 CPU Register Configuration



1-8. Two Index Registers (IX & IY). The two independent index registers hold a 16-bit base address that is used in indexed addressing modes. In this mode, an index register is used as a base to point to a region in memory from which data is to be stored or retrieved. An additional byte is included in indexed instructions to specify a displacement from this base. This displacement is specified as a two's complement signed integer. This mode of addressing greatly simplifies many types of programs, especiall where tables of data are used.

1-9. Interrupt Page Address Register (I). The Z80 CPU can be operated in a mode where an indirect call to any memory location can be achieved in response to an interrupt. The I Register is used for this purpose to store the high order 8-bits of the indirect address while the interrupting device provides the lower 8-bits of the address. This feature allows interrupt routines to be dynamically located anywhere in memory with absolute minimal access time to the routine.

1-10. Memory Refresh Register (R). The Z80 CPU contains a memory refresh counter to enable dynamic memories to be used with the same ease as static memories. This 7-bit register is automatically incremented after each instruction fetch. The data in the refresh counter is sent out on the lower portion of the address bus along with a refresh control signal while the CPU is decoding and executing the fetched instruction. This mode of refresh is totally transparent to the programmer and does not slow down the CPU operation. The programmer can load the R register for testing purposes, but this register is normally not used by the programmer.

1-11. ACCUMULATOR AND FLAG REGISTERS. The CPU includes two independent 8-bit accumulators and associated 8-bit flag registers. The accumulator holds the results of 8-bit arithmetic or logical operations while the flag register indicates specific conditions for 8 or 16-bit operations, such as indicating whether or not the result of an operation is equal to zero. The programmer selects the accumulator and flag pair that he wishes to work with a single exchange instruction so that he may easily work with either pair.

1-12. GENERAL PURPOSE REGISTERS. There are two matched sets of general purpose registers, each set containing six 8-bit registers that may be used individually as 8-bit registers or as 16-bit register pairs by the programmer. One set is called BC, DE and HL while the complementary set is called BC', DE' and HL'. At any one time the programmer can select either set of registers to work with through a single exchange command for the entire set. In systems where fast interrupt response is required, one set of general purpose registers and an accumulator/flag register may be reserved for handling this very fast routine. Only a simple exchange command need be executed to go between the routines. This greatly reduces interrupt service time by eliminating the requirement for saving and retrieving register contents in the external stack during interrupt or subroutine processing. These general purpose registers are used for a wide range of applications by the programmer. They also simplify programming, especially in ROM based systems where little external read/write memory is available.

1-13. ARITHMETIC & LOGIC UNIT (ALU)

1-14. The 8-bit arithmetic and logical instructions of the CPU are executed in the ALU. Internally the ALU communicates with the registers and the external data bus on the internal data bus. The type of functions performed by the ALU include:

Add

Subtract

Logical AND

Logical OR

Logical EXCLUSIVE OR

Compare

Left or right shifts or rotates (arithmetic and logical)

Increment

Decrement

Set bit

Reset bit

Test bit

1-15. INSTRUCTION REGISTER AND CPU CONTROL

1-16. As each instruction is fetched from memory, it is placed in the instruction register and decoded. The control section performs this function and then generates and supplies all of the control signals necessary to read or write data from or to the registers, controls the ALU and provides all required external control signals.

SECTION 2

Z80 INSTRUCTION SET

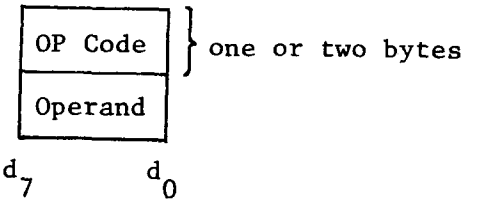
2-1. INTRODUCTION

2-2. The Z80 instruction set of 158 instructions can best be described by first discussing in general the main features. These features include its addressing modes; status and flags; interrupt modes; load and exchange instructions; block transfer and search instructions; arithmetic and logical instructions; rotate and shift instructions; bit manipulation instructions; jump, call, and return instructions; input/output instruction; and miscellaneous instructions. Included in the discussion of the addressing modes are descriptions of subroutines and subroutine use of the stack. Following this general description of the instruction set, specific information about the syntax of the assembly language is provided. Then each instruction is described in detail in alphabetical order.

2-3. INSTRUCTION SET FEATURES

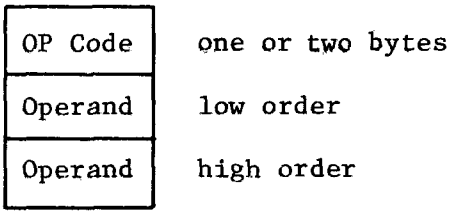
2-4. ADDRESSING MODES. Most of the Z80 instructions operate on data stored in internal CPU registers, external memory or in the I/O ports. Addressing refers to how the address of this data is generated in each instruction. The following paragraphs gives a brief summary of the types of addressing used in the Z80. Many instructions include more than one operand (such as arithmetic instructions or loads). In these cases, two types of addressing may be employed. For example, load can use immediate addressing to specify the source and register indirect or indexed addressing to specify the destination.

2-5. Immediate Addressing. In this mode of addressing the byte following the OP code in memory contains the actual operand.



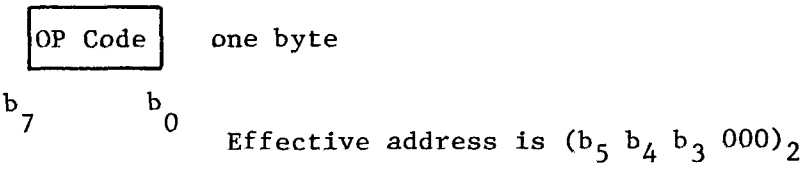
Examples of this type of instruction would be to load the accumulator with a constant, where the constant is the byte immediately following the OP code.

2-6. Immediate Extended Addressing. This mode is merely an extension of immediate addressing in that the two bytes following the OP code are the operand.

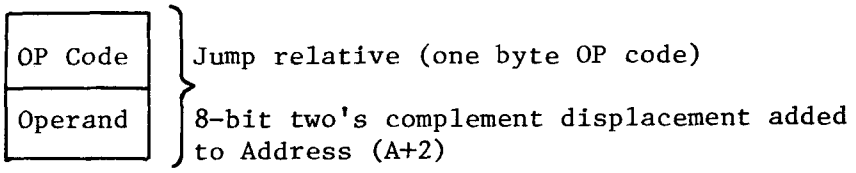


Examples of this type of instruction would be to load the HL register pair (16-bit register) with 16 bits (2 bytes) of data.

2-7. Modified Page Zero Addressing. The Z80 has a special single byte call instruction to any of 8 locations in page zero of memory. This instruction (which is referred to as a restart) sets the PC to an effective address in page zero. The value of this instruction is that it allows a single byte to specify a complete 16-bit address where commonly called subroutines are located, thus saving memory space.

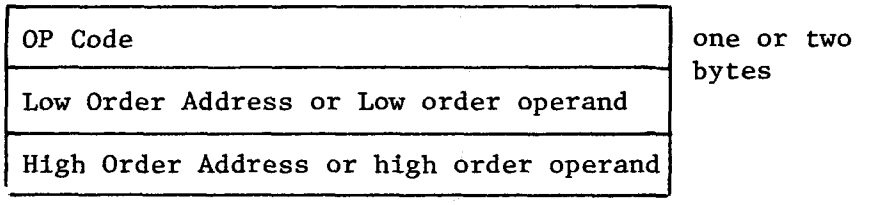


2-8. Relative Addressing. Relative addressing uses one byte of data following the OP code to specify a displacement from the existing program to which a program jump can occur. This displacement is a signed two's complement number that is added to the address of the OP code of the following instruction.



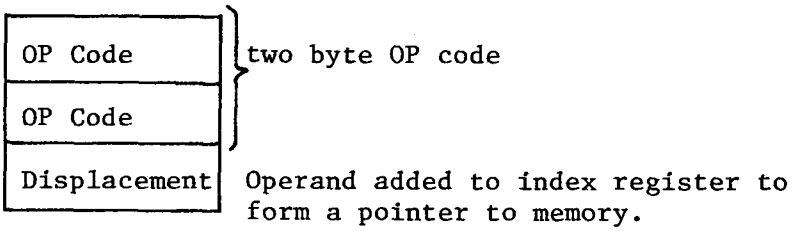
The value of relative addressing is that it allows jumps to nearby locations while only requiring two bytes of memory space. For most programs, relative jumps are by far the most prevalent type of jump due to the proximity of related program segments. Thus, these instructions can significantly reduce memory space requirements. The signed displacement can range between +127 and -128 from A + 2. This allows for a total displacement of +129 to -126 from the jump relative OP code address. Another major advantage is that it allows for relocatable code.

2-9. Extended Addressing. Extended Addressing provides for two bytes (16 bits) of address to be included in the instruction. This data can be an address to which a program can jump or it can be an address where an operand is located.



Extended addressing is required for a program to jump from any location in memory to any other location, or load and store data in any memory location. When extended addressing is used to specify the source or destination address of an operand, the notation (nn) will be used to indicate the content of memory at nn, where nn is the 16-bit address specified in the instruction. This means that the two bytes of address nn are used as a pointer to a memory location. The use of the parentheses always means that the value enclosed within them is used as a pointer to a memory location. For example, (1200) refers to the contents of memory at location 1200.

2-10. Indexed Addressing. In this type of addressing, the byte of data following the OP code contains a displacement which is added to one of the two index registers (the OP code specifies which index register is used) to form a pointer to memory. The contents of the index register are not altered by this operation.



2-11. An example of an indexed instruction would be to load the contents of the memory location (Index Register + Displacement) into the accumulator. The displacement is a signed two's complement number. Indexed addressing greatly simplifies programs using tables of data since the index register can point to the start of any table. Two index registers are provided since very often operations require two or more tables. Indexed addressing also allows for relocatable code.

2-12. The two index registers in the Z80 are referred to as IX and IY. To indicate indexed addressing the notation:

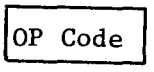
(IX+d) or (IY+d)

is used. Here d is the displacement specified after the OP code. The parentheses indicate that this value is used as a pointer to external memory.

2-13. Register Addressing. Many of the Z80 OP codes contain bits of information that specify which CPU register is to be used for an operation. An example of register addressing would be to load the data in register B into register C.

2-14. Implied Addressing. Implied addressing refers to operations where the OP code automatically implies one or more CPU registers as containing the operands. An example is the set of arithmetic operations where the accumulator is always implied to be the destination of the results.

2-15. Register Indirect Addressing. This type of addressing specifies a 16-bit CPU register pair (such as HL) to be used as a pointer to any location in memory. This type of instruction is very powerful and it is used in a wide range of applications.



one or two bytes

An example of this type of instruction would be to load the accumulator with the data in the memory location pointed to by the HL register contents. Indexed addressing is actually a form of register indirect addressing except that a displacement is added with indexed addressing. Register indirect addressing allows for very powerful but simple to implement memory accesses. The block move and search commands in the Z80 are extensions of this type of addressing where automatic register incrementing, decrementing and comparing has been added. The notation for indicating register indirect addressing is to put parentheses around the name of the register that is to be used as the pointer. For example, the symbol

(HL)

specifies that the contents of the HL register are to be used as a pointer to a memory location. Often register indirect addressing is used to specify 16-bit operands. In this case, the register contents point to the lower order portion of the operand while the register contents are automatically incremented to obtain the upper portion of the operand.

2-16. Bit Addressing. The Z80 contains a large number of bit set, reset and test instructions. These instructions allow any memory location or CPU register to be specified for a bit operation through one of three previous addressing modes (register, register indirect and indexed) while three bits in the OP code specify which of the eight bits is to be manipulated.

2-17. Stack Pointer Addressing. Memory locations may be addressed in the 16-bit stack pointer register (SP). There are two stack operations which may be performed:

1. PUSH, which puts data into a stack,

2. POP, which retrieves data from a stack.

Note that the stack area must reside in read/write memory. The stack pointer is initialized to the top location in the stack at the start of a program. In a stack operation a 16-bit register pair is transferred to or from the stack.

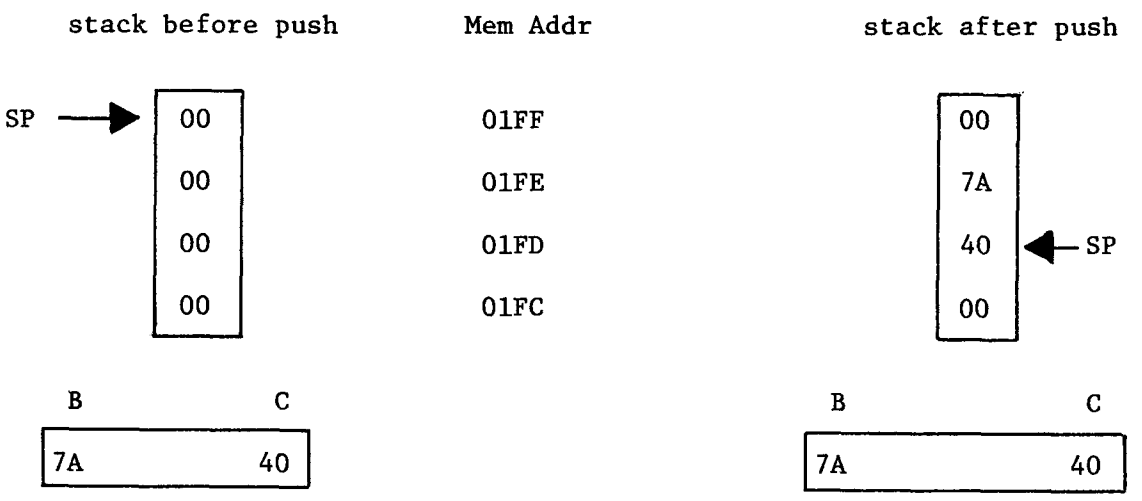
2-18. For the PUSH operation the contents of the register pair are transferred to the stack:

1. The most significant 8-bits of data are stored at the memory address less one than the contents of the stack pointer.

2. The least significant 8 bits of data are stored at the memory address less two than the contents of the stack pointer.

3. The stack pointer is automatically decremented by two.

PUSH BC

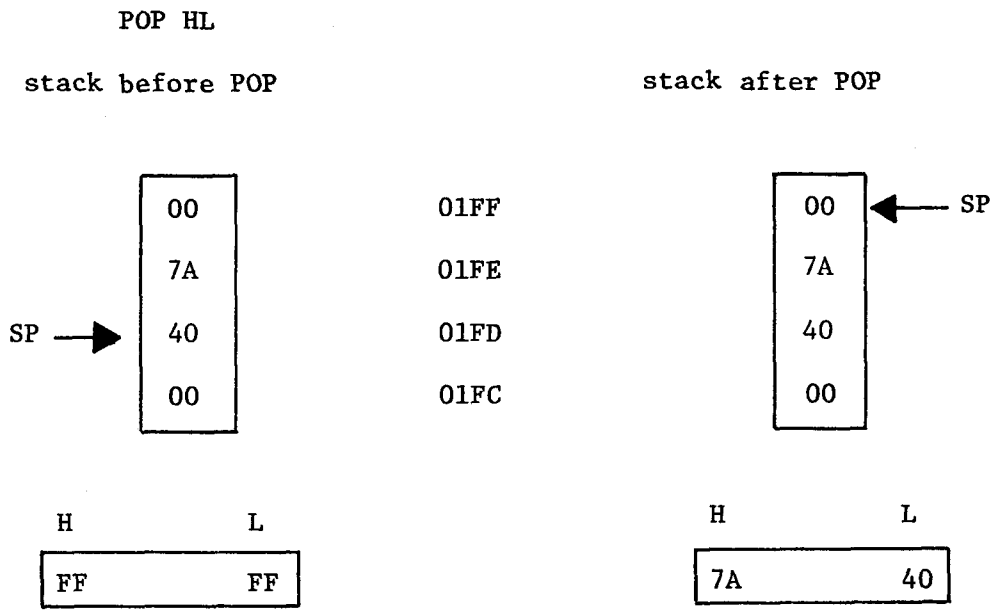


2-19. For the POP operation, 16 bits of data are taken from the stack and placed in the 16-bit register pair:

1. The second register of the pair (or the least significant byte of the pair) is loaded from the memory address held in the stack pointer.

2. The first register of the pair (or the most significant byte of the pair) is loaded from the memory address one greater than the address held in the stack pointer.

3. The stack pointer is automatically incremented by two.



2-20. Subroutine Addressing

2-21. Subroutines are blocks of instructions that can be called during the execution of a sequence of instructions. Subroutines can be called from main programs or from other subroutines. A subroutine is entered by the CALL opcode as in:

```
CALL REWIND
```

2-22. Parameters such as those used by the macros are not used with subroutines. When a call instruction is encountered during execution of a program, the PC is changed to the first instruction of the subroutine. The subsequent address of the invoking program is pushed on the stack. Control will return to this point when the subroutine is finished. The processor continues to execute the subroutine until it encounters a RET (return) instruction. At this point the return address is popped off the stack into the PC, and the processor returns to the address of the instruction following the CALL, to continue execution from that point.

2-23. Subroutines of any size can be invoked from programs or other subroutines of any size, without restriction. Care must be taken when nesting subroutines (subroutines within subroutines) that pushes and pops remain balanced at each level. If the processor encounters a RET with an un-popped push on the stack, the PC will be set to a meaningless address rather than to the next instruction following the CALL.

2-24. Tradeoffs must be considered between:

1. using a block of code repetitively in line, and

2. calling the block repetitively as a subroutine.

2-25. Program size can usually be saved by using the subroutine. If the repetitive block contains N bytes and it is repeated on M occasions in the program,

1. MxN bytes would be used in direct programming, while

2. 3M (for CALLS)

   + N (for the block)

   + 1 (for the RET)

   = 3M+N+1 bytes would be required if using a subroutine.

2-26. For example, for a block of 20 bytes used 5 times, in-line programming would require 100 bytes while a subroutine would require 36.

2-27. An added advantage of subroutines is that with careful naming, program structures become clearer, easier to read and easier to debug and maintain. Subroutines written for one purpose can be employed elsewhere in other programs requiring the same function.

2-28. Subroutines differ from Macros in several ways:

1. Subroutine code is assembled into an object program only once although it may be called many times. Macro code is assembled in line every place the macro is used.

2. Registers and pointers required by a subroutine must be set up before the CALL. No parameters are used and no argument string can be issued. Macros, through their use of parameters, can modify the settings of registers on each occurrence.

2-29. Subroutine Use Of The Stack. When a call to a subroutine is executed, the contents of the program counter are pushed onto the stack automatically. Recall that the program counter contains the next memory address to be executed. After the PC is pushed onto the stack, the starting address of the subroutine is placed into the PC and the branch to the subroutine is completed. At the end of the subroutine, a return instruction pops the address off the stack into the PC, and control is transferred to the memory address after the call. These operations are automatic when the CALL and RET instructions are executed.

2-30. Note that parameters can be passed to a subroutine because the stack and stack pointer can be manipulated and updated by special Z80 instructions.

2-31. The save type of operation as described for a subroutine also occurs for external interrupts monitored by the CPU.

2-32. Z80 STATUS INDICATORS (FLAGS)

2-33. Flag Register. The flag register (F and F') supplies information to the user regarding the status of the Z80 at any given time. The bit positions for each flag is shown below:

7	6	5	4	3	2	1	0
S	Z	X	H	X	P/V	N	C

WHERE:

C = CARRY FLAG
N = ADD/SUBTRACT FLAG
P/V = PARITY/OVERFLOW FLAG
H = HALF-CARRY FLAG
Z = ZERO FLAG
S = SIGN FLAG
X = NOT USED

Each of the two Z80 Flag Registers contains 6 bits of status information which are set or reset by CPU operations. (Bits 3 and 5 are not used.) Four of these bits are testable (C,P/V,Z and S) for use with conditional jump, call or return instructions. Two flags are not testable (H,N) and are used for BCD arithmetic.

2-34. Carry Flag (C). The carry bit is set or reset depending on the operation being performed. For ADD instructions that generate a carry, and SUBTRACT instructions that generate a borrow, the Carry Flag will be set. The Carry Flag is reset by an ADD that does not generate a carry and a SUBTRACT that does not generate a borrow. This saved carry facilitates software routines for extended precision arithmetic.
Also, the DAA instruction will set the Carry Flag if the conditions for making the decimal adjustment are met.

2-35. For instructions RLA, RRA, RL and RR , the carry bit is used as a link between the LSB and MSB for any register or memory location. During instructions RLCA, RLC s and SLA s, the carry contains the last value shifted out of bit 7 of any register or memory location. During instructions RRCA, RRC s, SRA s and SRL s the carry contains the last value shifted out of bit 0 of any register or memory location.

2-36. For the logical instructions AND s, OR s and XOR s, the carry will be reset.

2-37. The Carry Flag can also be set (SCF) and complemented (CCF).

2-38. Add/Subtract Flag (N). This flag is used by the decimal adjust accumulator instruction (DAA) to distinguish between ADD and SUBTRACT instructions. For all ADD instructions, N will be set to an 0. For all SUBTRACT instructions, N will be set to a 1.

2-39. Parity/Overflow Flag. This flag is set to a particular state depending on the operation being performed.

2-40. For arithmetic operations, this flag indicates an overflow condition when the result in the Accumulator is greater than the maximum possible number (+127) or is less than the minimum possible number (-128). This overflow condition can be determined by examining the sign bits of the operands.

2-41. For addition, operands with different signs will never cause overflow. When adding operands with like signs and the result has a different sign, the overflow flag is set. For example:

```
+120 = 0111  1000     ADDEND
+105 = 0110  1001     AUGEND
+225 = 1110  0001     (-95) SUM
```

The adding of the two numbers together has resulted in a number that exceeds +127 and the two positive operands cause a negative number (-95) which is incorrect. The overflow flag is therefore set.

2-42. For subtraction, overflow can occur for operands of unlike signs. Operands of like sign will never cause overflow. For example:

```
  +127   0111  1111     MINUEND
(-) -64  1100  0000     SUBTRAHEND
  +191   1011  1111     DIFFERENCE
```

The minuend sign has changed from a positive to a negative, giving an incorrect difference. Overflow is therefore set. Another method for predicting an overflow is to observe the carry into and out of the sign bit. If there is a carry in and no carry out, or if there is no carry in and a carry out, then overflow has occurred.

2-43. This flag is also used with logical operations and rotate instructions to indicate the parity of the result. The number of 1 bits in a byte are counted. If the total is odd, ODD parity (P=0) is flagged. If the total is even, EVEN parity is flagged (P=1).

2-44. During search instructions CPI, CPIR, CPD, and CPDR and block transfer instructions LDI, LDIR, LDD, and LDDR the P/V flag monitors the state of the byte count register (BC). When decrementing, the byte counter results in a zero value, the flag is reset to 0, otherwise the flag is a 1.

2-45. During LD A,I and LD A,R instructions, the P/V flag will be set with the contents of the interrupt enable flip-flop (IFF2) for storage or testing.

2-46. When inputting a byte from an I/O device, IN r,(C), the flag will be adjusted to indicate the parity of the data.

2-47. Half Carry Flag (H). The Half Carry Flag (H) will be set or reset depending on the carry and borrow status between bits 3 and 4 of an 8-bit arithmetic operation. This flag is used by the decimal adjust accumulator instruction DAA to correct the result of a packed BCD add or subtract operation. The H flag will be set (1) or reset (0) according to the following table:

H	ADD	SUBTRACT
1	There is a carry from Bit 3 to Bit 4	There is a borrow from bit 4
0	There is no carry from Bit 3 to Bit 4	There is no borrow from bit 4

2-48. Zero Flag (Z). The Zero Flag (Z) is set or reset if the result generated by the execution of certain instructions is a zero.

2-49. For 8-bit arithmetic and logical operations, the Z flag will be set to a 1 if the resulting byte in the Accumulator is zero. If the byte is not zero, the Z flag is reset to 0 .

2-50. For compare (search) instructions, the Z flag will be set to a '1' if a comparison is found between the value in the Accumulator and the memory location pointed to by the contents of the register pair HL.

2-51. When testing a bit in a register or memory location, the Z flag will contain the complemented state of the indicated bit (see Bit b,s).

2-52. When inputting or outputting a byte between a memory location and an I/O device (INI;IND;OUTI and OUTD), if the result of B-1 is zero, the Z flag is set, otherwise it is reset. Also for byte inputs from I/O devices using IN r,(C), the Z flag is set to indicate a zero byte input.

2-53. Sign Flag (S). The Sign Flag (S) stores the state of the most significant bit of the Accumulator (Bit 7). When the Z80 performs arithmetic operations on signed numbers, binary two's complement notation is used to represent and process numeric information. A positive number is identified by a 0 in bit 7. A negative number is identified by a 1 . The binary equivalent of the magnitude of a positive number is stored in bits 0 to 6 for a total range of from 0 to 127. A negative number is represented by the two's complement of the equivalent positive number. The total range for negative numbers is from -1 to -128.

2-54. When inputting a byte from an I/O device to a register, IN r, (C), the S flag will indicate either positive (S=0) or negative (S=1) data. The state of the four testable flags is specified as follows:

FLAG	ON CONDITION	OFF CONDITION
Carry	C	NC
Zero	Z	NZ
Sign	M (minus)	P (plus)
Parity	PE (even)	PO (odd)

2-55. INTERRUPTS. The purpose of an interrupt is to allow peripheral devices to suspend CPU operation in an orderly manner and force the CPU to start a peripheral service routine. Usually this service routine is involved with the exchange of data, or status and control information, between the CPU and the peripheral. Once the service routine is completed, the CPU returns to the operation from which it was interrupted.

2-56. Interrupt Types.

Non-Maskable

A non-maskable interrupt will be accepted at all times by the CPU. When this occurs, the CPU ignores the next instruction that it fetches and instead does a restart to location 0066H. Thus, it behaves exactly as if it had received a restart instruction but, it is to a location that is not one of the 8 software restart locations. A restart is merely a call to a specific address in page 0 of memory.

Maskable

The CPU can be programmed to respond to the maskable interrupt interrupt in any one of three possible modes.

Mode 0

This mode is identical to the 8080A interrupt response mode. With this mode, the interrupting device can place any instruction on the data bus and the CPU will execute it. Thus, the interrupting device provides the next instruction to be executed instead of

the memory. Often this will be a restart instruction since the interrupting device only needs to supply a single byte instruction. Alternatively, any other instruction such as a 3 byte call to any location in memory could be executed.

The number of clock cycles necessary to execute this instruction is 2 more than the normal number for the instruction. This occurs since the CPU automatically adds 2 wait states to an interrupt response cycle to allow sufficient time to implement an external daisy chain for priority control. After the application of RESET the CPU will automatically enter interrupt Mode 0.

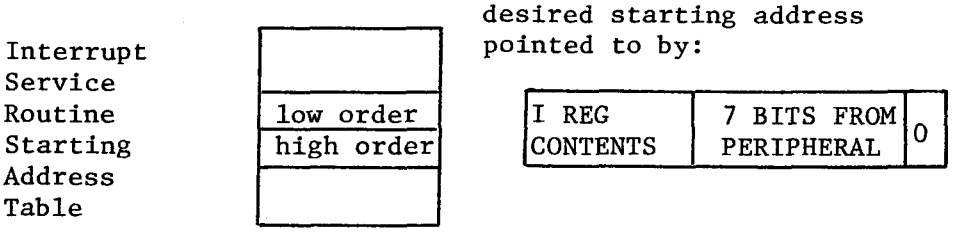
Mode 1

When this mode has been selected by the programmer, the CPU will respond to an interrupt by executing a restart to location 0038H. Thus the response is identical to that for a non-maskable interrupt except that the call location is 0038H instead of 0066H. Another difference is that the number of cycles required to complete the restart instruction is 2 more than normal due to the two added wait states.

Mode 2

This mode is the most powerful interrupt response mode. With a single 8 bit byte from the user, an indirect call can be made to any memory location.

With this mode the programmer maintains a table of 16-bit starting addresses for every interrupt service routine. This table may be located anywhere in memory. When an interrupt is accepted, a 16-bit pointer must be formed to obtain the desired interrupt service routine starting address from the table. The upper 8 bits of this pointer is formed from the contents of the I Register. The I Register must have been previously loaded with the desired value by the programmer, i.e. LDI, A. Note that a CPU reset clears the I register so that it is initialized to zero. The lower eight bits of the pointer must be supplied by the interrupting device. Actually only 7 bits are required from the interrupting device as the least significant bit must be a zero. This is required since the pointer is used to get two adjacent bytes to form a complete 16-bit service routine starting address, and the addresses must always start in even locations.



The first byte in the table is the least significant (low order) portion of the address. The programmer must obviously fill this table in with the desired addresses before any interrupts are to be accepted.

Note that this table can be changed at any time by the programmer (if it is stored in Read/Write Memory) to allow different peripherals to be serviced by different service routines.

Once the interrupting device supplies the lower portion of the pointer, the CPU automatically pushes the program counter onto the stack, obtains the starting address from the table and does a jump to this address. This mode of response requires 19 clock periods to complete (7 to fetch the lower 8 bits from the interrupting device, 6 to save the program counter, and 6 to obtain the jump address.)

Note that the Z80 peripheral devices all include a daisy chain priority interrupt structure that automatically supplies the programmed vector to the CPU during interrupt acknowledge. Refer to the Z80 PIO, Z80 SIO and Z80 CTC manuals for details.

2-57. Interrupt Enable/Disable.

2-58. The Z80 CPU has two interrupt inputs, a software maskable interrupt and a non-maskable interrupt. The non-maskable interrupt (NMI) can not be disabled by the programmer and it will be accepted whenever a peripheral device requests it. This interrupt is generally reserved for very important functions that must be serviced whenever they occur, such as an impending power failure. The maskable interrupt (INT) can be selectively enabled or disabled by the programmer. This allows the programmer to disable the interrupt during periods where his program has timing constraints that do not allow it to be interrupted. In the Z80 CPU there are enable flip flops (called IFF_1) and IFF_2 that are set or reset by the programmer using the Enabl Interrupt (EI) and Disable Interrupt (DI) instructions. When the IFF_1 is reset, an interrupt can not be accepted by the CPU. Table 2-1 summarizes the effect of the different instructions on the two enable flip flops.

2-59. There are two enable flip flops, called IFF_1 and IFF_2.

IFF_1	IFF_2
Actually disables interrupts from being accepted.	Temporary storage location for IFF_1.

The state of IFF_1 is used to actually inhibit interrupts while IFF_2 is used as a temporary storage location for IFF_1. The purpose of storing the IFF_1 will be subsequently explained.

2-60. A reset to the CPU will force both IFF_1 and IFF_2 to the reset state so that interrupts are disabled. They can then be enabled by an EI instruction at any time by the programmer. When an EI instruction is executed, any pending interrupt request will not be accepted until after the instruction following EI has been executed. This single instruction delay is necessary for cases when the following instruction is a return instruction and interrupts must not be allowed until the return has been completed. The EI instruction sets both IFF_1 and IFF_2 to the enable state. When an interrupt is accepted by the CPU, both IFF_1 and IFF_2 are automatically reset, inhibiting further interrupts until the programmer wishes to issue a new EI instruction. Note that for all of the previous cases, IFF_1 and IFF_2 are always equal.

2-61. The purpose of IFF_2 is to save the status of IFF_1 when a non-maskable interrupt occurs. When a non-maskable interrupt is accepted, IFF_1 is reset to prevent further interrupts until reenabled by the programmer. Thus, after a non-maskable interrupt has been accepted, maskable interrupts are disabled but the previous state of IFF_1 has been saved so that the complete state of the CPU just prior to the non-maskable interrupt can be restored at any time. When a Load Register A with Register (LD A,I) instruction or a Load Register A with Register R (LD A,R) instruction is executed, the state of IFF_2 is copied into the parity flag where it can be tested or stored.

2-62. A second method of restoring the status of IFF_1 is thru the execution of a Return from Non-Maskable Interrupt (RETN) instruction. Since this instruction indicates that the non-maskable interrupt service routine is complete, the contents of IFF_2 are now copied back into IFF_1, so that the status of IFF_1 just prior to the acceptance of the non-maskable interrupt will be restored automatically.

Table 2-1. Interrupt Enable/Disable Flip Flops

Action	IFF_1	IFF_2	
CPU Reset	0	0	
DI	0	0	
EI	1	1	
LD A,I	.	.	$IFF_2 \rightarrow$ Parity flag
LD A,R	.	.	$IFF_2 \rightarrow$ Parity flag
Accept NMI	0	.	
RETN	IFF_2	.	$IFF_2 \rightarrow IFF_1$
RETI	.	.	
Accept INT	0	0	

"." indicates no change

2-63. LOAD AND EXCHANGE INSTRUCTIONS. These instructions move data to and from registers, such as load B from D, load C from memory, store HL into memory, push IX into stack, and exchange AF with A'F'.

2-64. BLOCK TRANSFER AND SEARCH INSTRUCTIONS. This group includes several useful instructions.

2-65. The load and increment instruction moves one byte of data from memory pointed to by HL to another memory location pointed to by DE Both register pairs are automatically incremented and the byte counter (BC) is decremented. This instruction is extremely valuable in moving blocks of data.

2-66. Another instruction repeats the load and increment instruction automatically until the byte counter reaches zero. Thus, in one instruction, a block of data, up to 64K bytes in length, can be moved anywhere in memory.

2-67. The compare and increment instruction, compares the contents of the accumulator with that of memory pointed to by HL. The appropriate flag bits are set, HL is automatically incremented, and the byte counter is decremented.

2-68. The compare, increment, and repeat instruction repeats the above instruction until either a match is found or the counter reaches zero.

2-69. ARITHMETIC AND LOGICAL INSTRUCTIONS. These instructions include all the adds and subtracts, increments, compares, exclusive-ors, etc. The Z80 features the indexed addressing mode and double precision add with carry and subtract with carry.

2-70. ROTATE AND SHIFT INSTRUCTIONS. The Z80 includes four rotate accumulator instructions and logical shifts and arithmetic shifts. There are also two rotate digit instructions which are applicable to BCD arithmetic. With these a digit (4 bits) can be rotated with two digits in a memory location.

2-71. BIT MANIPULATION INSTRUCTIONS. There are three basic bit manipulation operations; test bit, set bit, and reset bit.

2-72. JUMP, CALL, AND RETURN. The Z80 has numerous conditional and unconditional jumps, calls, and returns. In addition, the Z80 has several jump relative instructions using relative addressing.

2-73. INPUT/OUTPUT INSTRUCTIONS.

2-74. The Z80 allows for a standard common I/O routine for all devices by including IO instructions that use the C register to contain the IO device address. Therefore one IO routine can be used with the device address placed in register C before entering the routine. Also instead of being restricted to inputting or outputting to and from the accumulator only, any register can be used.

2-75. The Z80 has eight block transfer IO instructions which are similar to the memory block transfer instructions. HL is the memory pointer, C is the device pointer, and B is the byte counter. Therefore, an IO block transfer can handle up to 256 bytes. Essentially these commands are a processor implementation of direct memory access (DMA), invoke by a software sequence.

2-76. MISCELLANEOUS FEATURES. The Z80 instruction set also includes a no-operation instruction.

2-77. Z80 ASSEMBLY LANGUAGE SYNTAX.

2-78. INTRODUCTION.

2-79. The assembly language of the Z80 is designed to minimize the number of different opcodes corresponding to the set of basic machine operations and to provide for a consistent description of instruction operands. The nomenclature has been defined with special emphasis on mnemonic value and readability.

2-80. An assembly language program, or source program, consists of statements in a sequence which defines the user's program. The statements consist of:

1. labels,
2. opcodes or pseudo-ops
3. operands, and
4. comments.

2-81. Certain rules define how assembly language statements are to appear. A statement has four separate and distinct parts or fields.

	LABEL	OPCODE	OPERANDS	COMMENT
ex:	LOOP:	LD	HL,VALUE	;GET VALUE

2-82. The first field is the LABEL field. The label is a name used to reference the program counter, another label, or a constant.

2-83. The second field is the OPCODE field. It specifies the operation to be performed. There are 74 Z80 opcodes and several pseudo-ops that are standard for the Z80. The standard pseudo-ops are described in Appendix B.

2-84. The third field is the OPERAND field. It provides address or data information for the OPCODE field. There may be zero or more operands in the operand field depending on the requirements of the opcode field.

2-85. The fourth field is the COMMENT field. It is used to document a program. The comment field may appear in a statement without the other fields. Comments are ignored by an assembler, but they are printed in the assembly listing.

2-86. Each of the above parts, or field, must be separated from each other by one or more commas or blanks. If more than one operand appears, they must be separated from each other by one or more commas.

2-87. LABELS

2-88. A label is a symbol representing up to 16 bits of information and is used to specify an address or data. By using labels effectively, the user can write assembly language programs more rapidly and make fewer errors.

2-89. A label is composed of one or more characters. If more than 6 characters are used for the label, only the first 6 will be recognized by a standard assembler. The first character of a label must not be a number (0-9) or a restricted character. The remaining characters cannot include a restricted character. The restricted characters are:

Control characters (0-2FH,7FH)

Space

' ( ) * + , - . /

: ; < = >

Note that a single dollar sign ($) is reserved to represent the program counter.

2-90. A label can start in any column if followed by a colon (:). It does not require a colon if started in column one.

2-91. OPCODES. The bulk of this manual describes the Z80 opcodes. Opcodes are 2 to 4 characters long and describe Z80 instructions.

2-92. STANDARD OPERANDS. There may be zero or more operands present in a statement depending upon the opcode used. An operand which appears in a statement may take one of the following forms.

1. A generic operand, such as the letter A, which stands for the accumulator.

2. A constant. The constant must be in the range 0 through OFFFFH. It can be in the following forms:

Decimal - Any number may be denoted as decimal by following it with the letter 'D'. E.g., 35, 249D. However, the assembler will consider any number which is undesignated as decimal.

Hexadecimal - must begin with a number (0-9) and end with the letter 'H'. E.g., 0AF1H

Octal - must end with the letter 'Q' or 'O'. E.g., 377Q, 2770

Binary - must end with the letter 'B'. E.g., 0110111B

ASCII - letters enclosed in quote marks will be converted to their ASCII equivalent value. E.g., 'A' = 41H

3. A label which appears elsewhere in the program. Note that labels cannot be defined by labels which have not yet appeared in the user program for 2-pass assemblers.

E.g.:

```
L EQU H

H EQU I

I EQU 7     IS NOT ALLOWED.


I EQU 7

H EQU I

L EQU H     IS ALLOWED.
```

4. The symbol $ is used to represent the value of the program counter of the current instruction.

5. Expressions. Expression evaluation capability is a function of the features of a particular assembler. In general, arithmetic and logical expressions are allowed, and parentheses may be used to assure correct evaluation.

2-93. OPERAND NOTATION. The following notation is used in the assembly language:

1) r specifies any one of the following registers: A,B,C,D,E,H,L.

2) (HL) specifies the contents of memory at the location addressed by the contents of the register pair HL.

3) n specifies a one-byte expression in the range (0 to 255). nn specifies a two-byte expression in the range (0 to 65535).

4) d specifies a one-byte expression in the range (-128,127).

5) (nn) specifies the contents of memory at the location addressed by the two-byte expression nn.

6) b specifies an expression in the range (0,7).

7) e specifies a one-byte expression in the range (-126,129).

8) cc specifies the state of the flags for conditional JR and JP instructions.

9) qq specifies any one of the register pairs BC, DE, HL or AF.

10) ss specifies any one of the following register pairs: BC, DE, HL, SP.

11) pp specifies any one of the following register pairs: BC, DE, IX, SP.

12) rr specifies any one of the following register pairs: BC, DE, IY, SP.

13) s specifies any of r, n, (HL, (IX+d), (IY+d).

14) dd specifies any one of the following register pairs: BC, DE, HL, SP.

15) m specifies any of r, (HL), (IX+d), (IY+d).

Note The enclosing of an expression wholly in parentheses indicates a memory address. The contents of the memory address equivalent to the expression value will be used as the operand value.

2-94. In doing relative addressing, the current value of the program counter must be subtracted from the label if a branch is to be made to that label address. E.g.:

```
JR  NC,LOOP-$

        ...will jump relative to 'LOOP'.
```

2-95. COMMENTS. A comment is defined as any string of characters following a semicolon. Comments are ignored by an assembler, but they are printed on the assembly listing. Comments can begin in any column:

```
;

;  this is a comment

;
```

2-96. UPPER/LOWER CASE.

Note The MOSTEK assembler and text editor allow the use of lower case letters for labels and comments.

2-97. OPCODES - DETAILED DESCRIPTIONS.

2-98. INTRODUCTION. This section describes each Z80 opcode (instruction) in detail. The opcodes are presented in alphabetical order, one per page. Each instruction is introduced by its mnemonic opcode and symbolic operands. Then follows a brief description, operation, valid operand combinations, machine code, detailed description, condition bits affected, and one or more examples.

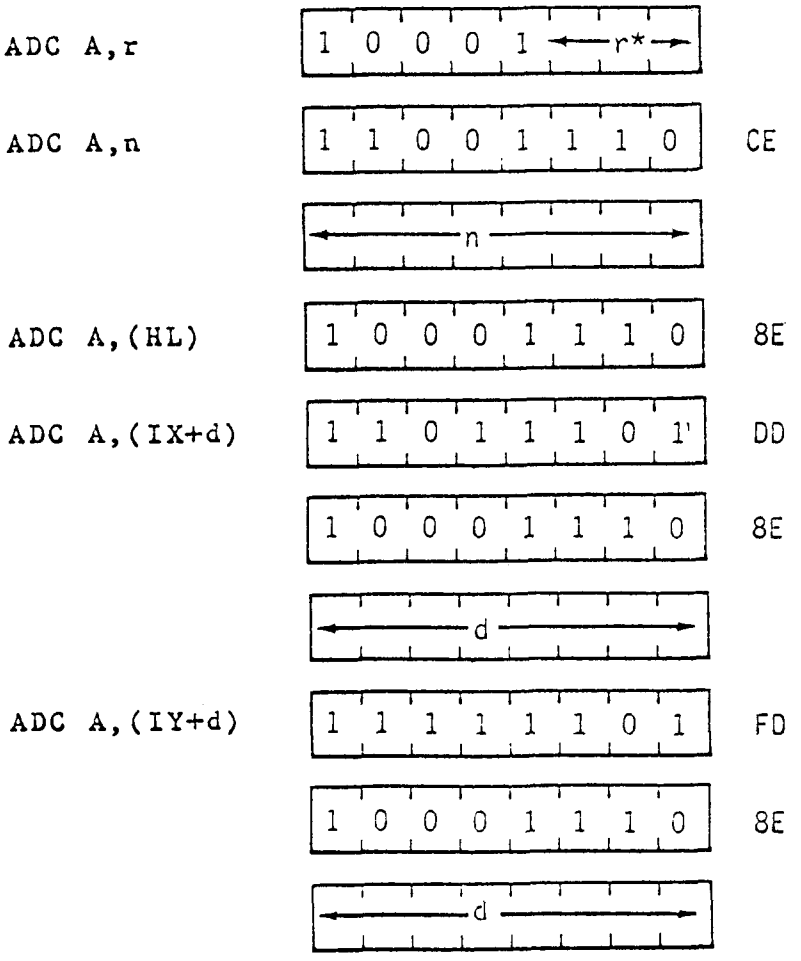
ADC A, s

Operation. A ← A + s + CY

Format:

Opcode	Operands
ADC	A,s

The s operand is any of r,n,(HL),(IX+d) or (IY+d) as defined for the analogous ADD instruction. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B,C,D,E,H,L or A assembled as follows in the object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

The s operand, along with the Carry Flag ("C" in the F register) is added to the contents of the Accumulator, and the result is stored in the Accumulator.

INSTRUCTION	M CYCLES	T STATES
ADC A,r	1	4
ADC A,n	2	7(4,3)
ADC A,(HL)	2	7(4,3)
ADC A,(IX+d)	5	19(4,4,3,5,3)
ADC A,(IY+d)	5	19(4,4,3,5,3)

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from Bit 3; reset otherwise
P/V: Set if overflow; reset otherwise
N: Reset
C: Set if carry from Bit 7; reset otherwise

Example:

If the Accumulator contains 16H, the Carry Flag is set, the HL register pair contains 6666H, and address 6666H contains 10H, after the execution of

```
ADC A,(HL)
```

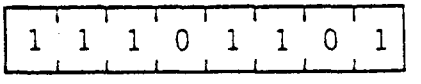
the Accumulator will contain 27H.

ADC HL, ss

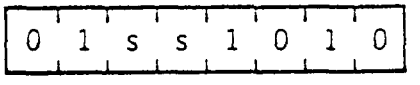
Operation: HL←HL+ss+CY

Format:

Opcode	Operands
ADC	HL,ss



ED



Description:

The contents of register pair ss (any of register pairs BC,DE,HL or SP) are added with the Carry Flag (C flag in the F register) to the contents of register pair HL, and the result is stored in HL. Operand ss is specified as follows in the assembled object code.

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

M CYCLES: 4 T STATES: 15(4,4,4,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if carry out of Bit 11; reset otherwise
P/V:	Set if overflow; reset otherwise
N:	Reset
C:	Set if carry from Bit 15; reset otherwise

Example:

If the register pair BC contains 2222H, register pair HL contains 5437H and the Carry Flag is set, after the execution of

```
ADC HL,BC
```

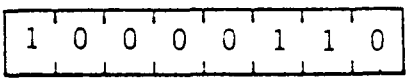
the contents of HL will be 765AH.

ADD A, (HL)

Operation: A ← A + (HL)

Format:

Opcode	Operands
ADD	A,(HL)



86

Description:

The byte at the memory address specified by the contents of the HL register pair is added to the contents of the Accumulator and the result is stored in the Accumulator.

M CYCLES: 2 T STATES: 7(4,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if carry from Bit 3; reset otherwise
P/V:	Set if overflow; reset otherwise
N:	Reset
C:	Set if carry from Bit 7; reset otherwise

Example:

If the contents of the Accumulator are A0H, and the content of the register pair HL is 2323H, and memory location 2323H contains byte 08H, after the execution of

ADD A,(HL)

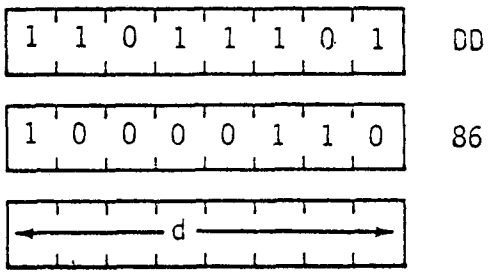
the Accumulator will contain A8H.

ADD A, (IX+d)

Operation: A ← A + (IX+d)

Format:

Opcode	Operands
ADD	A,(IX+d)



Description:

The contents of the Index Register (register pair IX) is added to a displacement d to point to an address in memory. The contents of this address is then added to the contents of the Accumulator and the result is stored in the Accumulator.

M CYCLES: 5 T STATES: 19(4,4,3,5,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if carry from Bit 3; reset otherwise
P/V:	Set if overflow; reset otherwise
N:	Reset
C:	Set if carry from Bit 7; reset otherwise

Example:

If the Accumulator contents are 11H, the Index Register IX contains 1000H, and if the content of memory location

1005H is 22H, after the execution of

ADD A,(IX+5H)

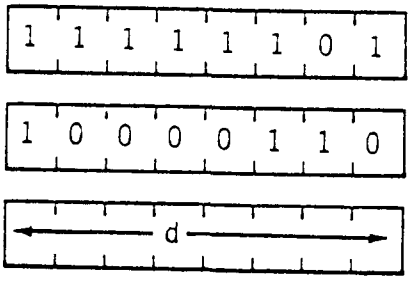
the contents of the Accumulator will be 33H.

ADD A, (IY+d)

Operation A ← A+(IY+d)

Format:

Opcode	Operands
ADD	A,(IY+d)



Description:

The contents of the Index Register (register pair IY) is added to a displacement d to point to an address in memory. The contents of this address is then added to the contents of the Accumulator and the result is stored in the Accumulator.

M CYCLES: 5 T STATES: 19(4,4,3,5,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if carry from Bit 3; reset otherwise
P/V:	Set if overflow; reset otherwise
N:	Reset
C:	Set if carry from bit 7; reset otherwise

Example:

If the Accumulator contents are 11H, the Index Register pair IY contains 1000H, and if the content of memory

location 1005H is 22H, after the execution of

```
ADD A,(IY+5H)
```

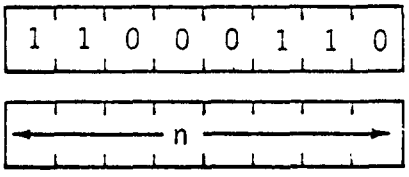
the contents of the Accumulator will be 33H.

ADD A, n

Operation: A ← A + n

Format.

Opcode	Operands
ADD	A,n



Description:

The integer n is added to the contents of the Accumulator and the results are stored in the Accumulator.

M CYCLES: 2 T STATES: 7(4,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if carry from Bit 3; reset otherwise
P/V:	Set if overflow; reset otherwise
N:	Reset
C:	Set if carry from Bit 7; reset otherwise

Example:

If the contents of the Accumulator are 23H, after the execution of

```
ADD A,33H
```

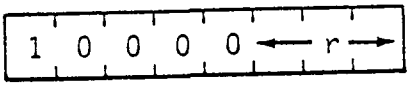
the contents of the Accumulator will be 56H.

ADD A, r

Operation: A ← A + r

Format:

Opcode	Operands
ADD	A,r



Description:

The contents of register r are added to the contents of the Accumulator, and the result is stored in the Accumulator. The symbol r identifies the registers A,B,C,D,E,H or L assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M CYCLES: 1 T STATES: 4

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if carry from Bit 3; reset otherwise
P/V:	Set if overflow; reset otherwise
N:	Reset
C:	Set if carry from Bit 7; reset otherwise

Example:

If the contents of the Accumulator are 44H, and the contents of register C are 11H, after the execution of

```
ADD A,C
```

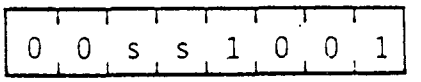
the contents of the Accumulator will be 55H.

ADD HL, ss

Operation: HL ← HL+ss

Format:

Opcode	Operands
ADD	HL,ss



Description:

The contents of register pair ss (any of register pairs BC, DE, HL or SP) are added to the contents of register pair HL and the result is stored in HL. Operand ss is specified as follows in the assembled object code.

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

M CYCLES: 3 T STATES: 11(4,4,3)

Condition Bits Affected:

S:	Not affected
Z:	Not affected
H:	Set if carry out of Bit 11; reset otherwise
P/V:	Not affected
N:	Reset
C:	Set if carry from Bit 15; reset otherwise

Example:

If register pair HL contains the integer 4242H and register pair DE contains 1111H, after the execution of

```
ADD HL,DE
```

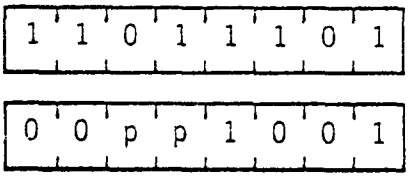
the HL register pair will contain 5353H.

ADD IX, pp

Operation: IX ← IX + pp

Format:

Opcode	Operands
ADD	IX,pp



Description:

The contents of register pair pp (any of register pairs BC,DE,IX or SP) are added to the contents of the Index Register IX, and the results are stored in IX. Operand pp is specified as follows in the assembled object code.

Register Pair	pp
BC	00
DE	01
IX	10
SP	11

M CYCLES: 4 T STATES: 15(4,4,4,3)

Condition Bits Affected:

S:	Not affected
Z:	Not affected
H:	Set if carry out of Bit 11; reset otherwise
P/V:	Not affected
N:	Reset
C:	Set if carry from Bit 15; reset otherwise

Example:

If the contents of Index Register IX are 333H and the contents of register pair BC are 5555H, after the execution of

```
ADD IX,BC
```

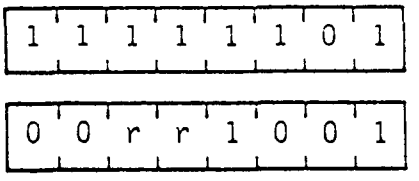
the contents of IX will be 8888H.

ADD IY, rr

Operation: IY←IY+rr

Format:

Opcode	Operands
ADD	IY,rr



Description:

The contents of register pair rr (any of register pairs BC,DE,IY or SP) are added to the contents of Index Register IY, and the result is stored in IY. Operand rr is specified as follows in the assembled object code.

Register Pair	rr
BC	00
DE	01
IY	10
SP	11

M CYCLES: 4 T STATES: 15(4,4,4,3)

Condition Bits Affected:

S:	Not affected
Z:	Not affected
H:	Set if carry out of Bit 11; reset otherwise
P/V:	Not affected
N:	Reset
C:	Set if carry from Bit 15; reset otherwise

Example:

If the contents of Index Register IY are 333H and the contents of register pair BC are 555H, after the execution of

```
ADD IY,BC
```

the contents of IY will be 8888H.

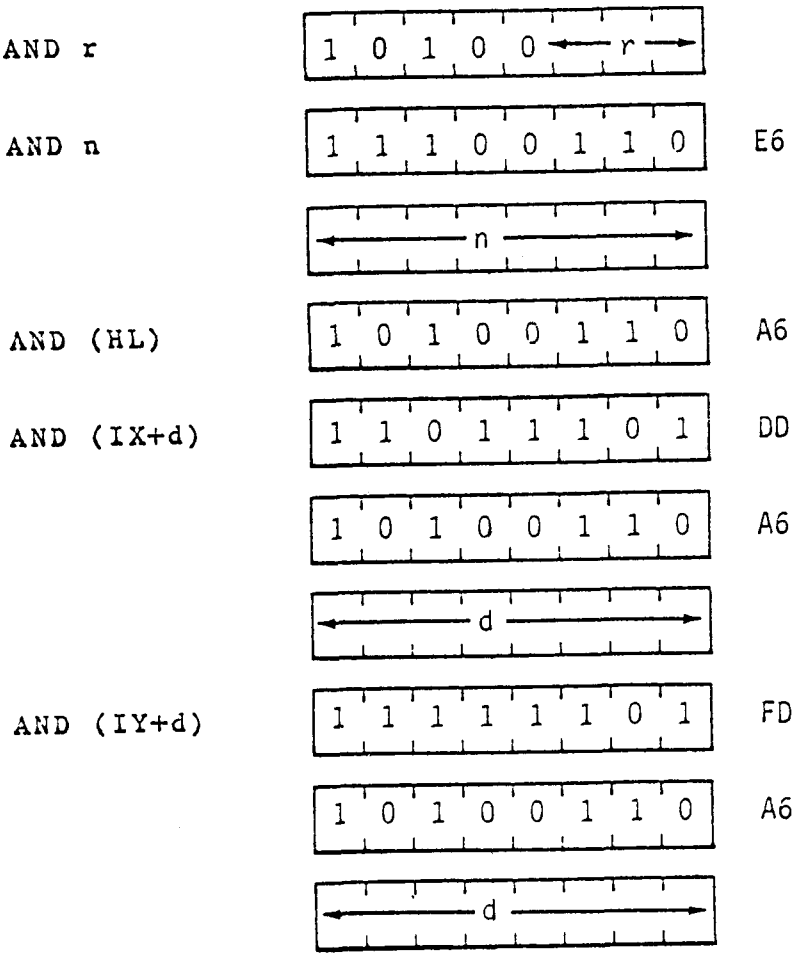
AND s

Operation: $A \leftarrow A \wedge s$

Format:

Opcode	Operands
AND	s

The s operand is any of r,n,(HL),(IX+d) or (IY+d), as defined for the analogous ADD instructions. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B,C,D,E,H,L or A assembled as follows in the object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

A logical AND operation, bit by bit, is performed between the byte specified by the s operand and the byte contained in the Accumulator; the result is stored in the Accumulator.

INSTRUCTION	M CYCLES	T STATES
AND r	1	4
AND n	2	7(4,3)
AND (HL)	2	7(4,3)
AND (IX+d)	5	19(4,4,3,5,3)
AND (IX+d)	5	19(4,4,3,5,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set
P/V:	Set if parity even; reset otherwise
N:	Reset
C:	Reset

Example:

If the B register contains 7BH (01111011) and the Accumulator contains C3H (11000011) after the execution of

AND B

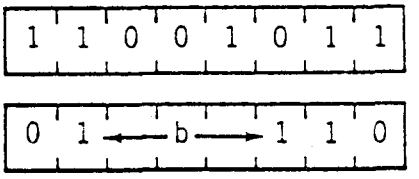
the Accumulator will contain 43H (01000011).

BIT b, (HL)

Operation: $Z \leftarrow \overline{(HL)_b}$

Format:

Opcode	Operands
BIT	b,(HL)



Description:

After the execution of this instruction, the Z flag in the F register will contain the complement of the indicated bit within the contents of the HL register pair. Operand b is specified as follows in the assembled object code:

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

M CYCLES: 3 T STATES: 12(4,4,4)

Condition Bits Affected:

S:	Unknown
Z:	Set if specified Bit is 0; reset otherwise
H:	Set
P/V:	Unknown
H:	Reset
C:	Not affected

Example:

If the HL register pair contains 4444H, and bit 4 in the memory location 444H contains 1, after the execution of

```
BIT 4, (HL)
```

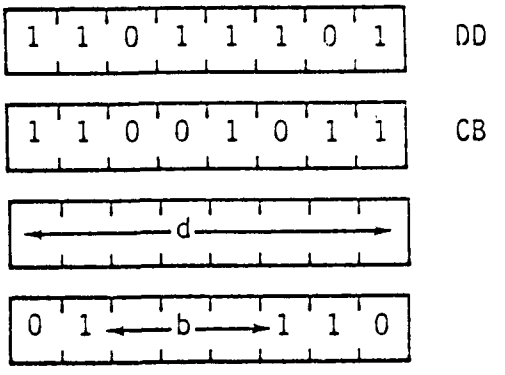
the Z flag in the F register will contain 0, and bit 4 in memory location 4444H will still contain 1. (Bit 0 in memory location 4444H is the least significant bit.)

BIT b, (IX+d)

Operation: $Z \leftarrow \overline{(IX+d)}_b$

Format:

Opcode	Operands
BIT	b,(IX+d)



Description:

After the execution of this instruction, the Z flag in the F register will contain the complement of the indicated bit within the contents of the memory location pointed to by the sum of the contents register pair IX (Index Register IX) and the two's complement displacement integer d. Operand b is specified as follows in the assembled object code.

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

M CYCLES: 5 T STATES: 20(4,4,3,5,4)

Condition Bits Affected:

S:	Unknown
Z:	Set if specified Bit is 0; reset otherwise
H:	Set
P/V:	Unknown
N:	Reset
C:	Not affected

Example:

If the contents of Index Register IX are 2000H, and bit 6 in memory location 2004H contains 1, after the execution of

```
BIT 6, (IX+4H)
```

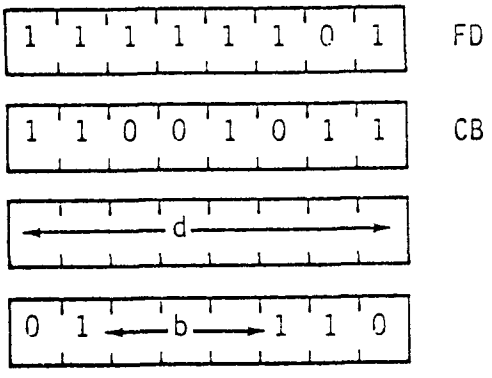
the Z flag in the F register will contain 0, and bit 6 in memory location 2004H will still contain 1. (Bit 0 in memory location 2004H is the least significant bit.)

BIT b, (IY+d)

Operation: $Z \leftarrow \overline{(IY+d)_b}$

Format:

Opcode	Operands
BIT	b,(IY+d)



Description:

After the execution of this instruction, the Z flag in the F register will contain the complement of the indicated bit within the contents of the memory location pointed to by the sum of the contents of register pair IY (Index Register IY) and the two's complement displacement integer d. Operand b is specified as follows in the assembled object code:

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

M CYCLES: 5 T STATES: 20(4,4,3,5,4)

Condition Bits Affected:

S:	Unknown
Z:	Set if specified Bit is 0; reset otherwise
H:	Set
P/V:	Unknown
N:	Reset
C:	Not affected

Example:

If the contents of Index Register are 2000H, and bit 6 in memory location 2004H contains 1, after the execution of

```
BIT 6, (IY+4H)
```

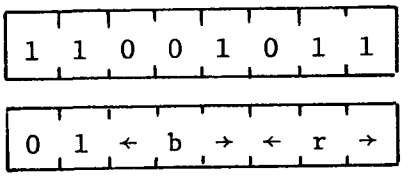
the Z flag in the F register sill contain 0, and bit 6 in memory location 2004H will still contain 1. (Bit 0 in memory location 2004H is the least significant bit.)

BIT b, r

Operation: $Z \leftarrow \overline{r}_b$

Format:

Opcode	Operands
BIT	b, r



Description:

After the execution of this instruction, the Z flag in the F register will contain the complement of the indicated bit within the contents of the r - register. Operands b and r are specified as follows in the assembled object code:

Bit Tested	b	register	r
0	000	B	000
1	001	C	001
2	010	D	010
3	011	E	011
4	100	H	100
5	101	L	101
6	110	A	111
7	111		

M CYCLES: 2 T STATES: 8 (4,4)

Condition Bits Affected:

S:	Unknown
Z:	Set if specified Bit is 0; reset otherwise
H:	Set
P/V:	Unknown
H:	Reset
C:	Not affected

Example:

If bit 4 in the B-register contains 1, after the execution of

```
BIT 4, B
```

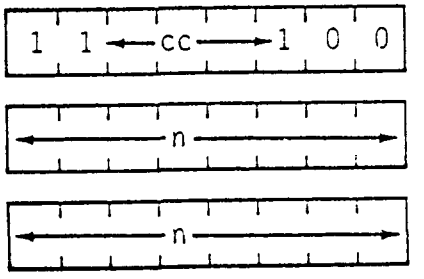
the Z flag in the F register will contain 0, and bit 4 in the B register will still contain 1. (Bit 0 in the B-register is the least significant bit.)

CALL cc, nn

Operation: IF cc TRUE: $(SP\text{-}1) \leftarrow PC_H$
$(SP\text{-}2) \leftarrow PC_L$, $PC \leftarrow nn$

Format:

Opcode	Operands
CALL	cc,nn



Note: The first of the two n operands in the assembled object code above is the least significant byte of the two-byte memory address.

Description:

If condition cc is true, this instruction pushes the current contents of the Program Counter (PC) onto the top of the external memory stack, then loads the operands nn into PC to point to the address in memory where the first opcode of a subroutine is to be fetched. (At the end of the subroutine, a RETurn instruction can be used to return to the original program flow by popping the top of the stack back into PC.) If condition cc is false, the Program Counter is incremented as usual, and the program continues with the next sequential instruction. The stack push is accomplished by first decrementing the current contents of the Stack Pointer (SP), loading the high-order byte of the PC contents into the memory address now pointed to by SP; then decrementing SP again, and loading the low-order byte of the PC contents into the top of the stack. Note: Because this is a 3-byte instruction, the Program Counter will have been incremented by 3 before

the push is executed. Condition cc is programmed as one of eight status which corresponds to condition bits in the Flag Register (register F). These eight status are defined in the table below, which also specifies the corresponding cc bit fields in the assembled object code:

cc	Condition	Relevant Flag
000	NZ non zero	Z
001	Z zero	Z
010	NC non carry	C
011	C carry	C
100	PO parity odd	P/V
101	PE parity even	P/V
110	P sign positive	S
111	M sign negative	S

If cc is true:

M CYCLES: 5 T STATES: 17(4,3,4,3,3)

If cc is false:

M CYCLES: 3 T STATES: 10(4,3,3)

Condition Bits Affected: None

Example:

If the C Flag in the F register is reset, the contents of the Program Counter are 1A47H, the contents of the Stack Pointer are 3002H, and memory locations have the contents:

Location	Contents
1A47H	D4H
1A48H	35H
1A49H	21H

then if an instruction fetch sequence begins, the three-byte instruction D43521H will be fetched to the CPU for execution. The mnemonic equivalent of this is

CALL NC,2135H

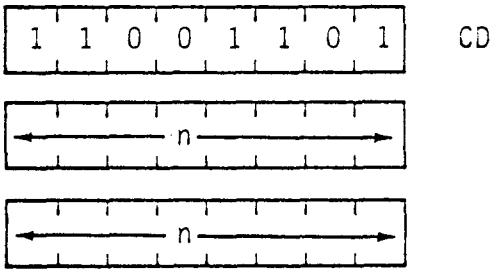
After the execution of this instruction, the contents of memory address 3001H will be 1AH, the contents of address 3000H will be 4AH, the contents of the Stack Pointer will be 3000H, and the contents of the Program Counter will be 2135H, pointing to the address of the first opcode of the subroutine now to be executed.

CALL nn

Operation: $(SP-1) \leftarrow PC_H$, $(SP-2) \leftarrow PC_L$, $PC \leftarrow nn$

Format:

Opcode	Operands
CALL	nn



Note: The first of the two n operands in the assembled object code above is the least significant byte of a two-byte memory address.

Description:

After pushing the current contents of the Program Counter (PC) onto the top of the external memory stack, the operands nn are loaded into PC to point to the address in memory where the first opcode of a subroutine is to be fetched. (At the end of the subroutine, a RETurn instruction can be used to return to the original program flow by popping the top of the stack back into PC.) The push is accomplished by first decrementing the current contents of the Stack Pointer (register pair SP), loading the high-order byte of the PC contents into the memory address now pointed to by the SP; then decrementing SP again, and loading the low-order byte of the PC contents into the top of stack. Note: Because this is a 3-byte instruction, the Program Counter will have been incremented by 3 before the push is executed.

M CYCLES: 5 T STATES: 17(4,3,4,3,3)

Condition Bits Affected: None

Example:

If the contents of the Program Counter are 1A47H, the contents of the Stack Pointer are 3002H, and memory locations have the contents:

Location	Contents
1A47H	CDH
1A48H	35H
1A49H	21H

then if an instruction fetch sequence begins, the three-byte instruction CD3521H will be fetched to the CPU for execution. The mnemonic equivalent of this is

CALL 2135H

After the execution of this instruction, the contents of memory address 3001H will be 1AH, the contents of address 3000H will be 4AH, the contents of the Stack Pointer will be 3000H, and the contents of the Program Counter will be 2135H, pointing to the address of the first opcode of the subroutine now to be executed.

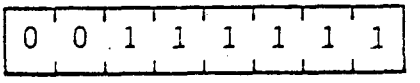
CCF

Operation: $CY \leftarrow \overline{CY}$

Format:

Opcode

CCF



3F

Description:

The C flag in the F register is inverted.

M CYCLES: 1 T STATES: 4

Condition Bits Affected:

S:	Not affected
Z:	Not affected
H:	Previous carry will be copied
P/V:	Not affected
N:	Reset
C:	Set if CY was 0 before operation; reset otherwise

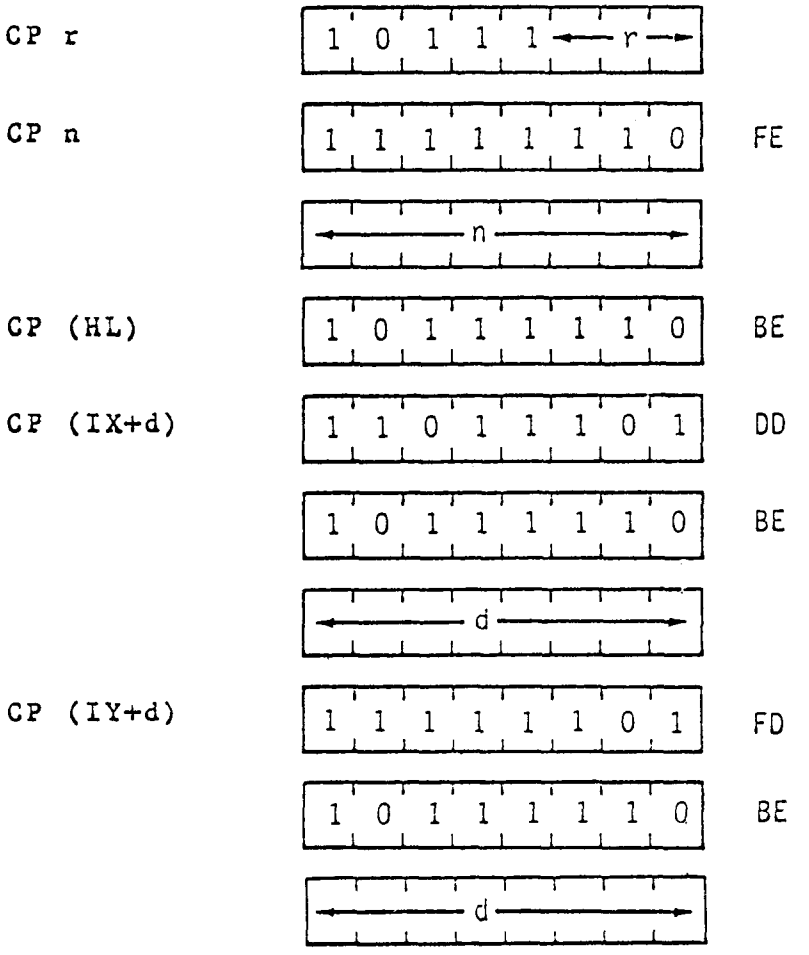
CP s

Operation: A - s

Format:

Opcode	Operands
CP	s

The s operand is any of r,n,(HL),(IX+d) or (IY+d), as defined for the analogous ADD instructions. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B,C,D,E,H,L or A assembled as follows in the object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

The contents of the s operand are compared with the contents of the Accumulator. If there is a true compare, a flag is set.

INSTRUCTION	M CYCLES	T STATES
CP r	1	4
CP n	2	7(4,3)
CP (HL)	2	7(4,3)
CP (IX+d)	5	19(4,4,3,5,3)
CP (IY+d)	5	19(4,4,3,5,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if borrow from Bit 4; reset otherwise
P/V:	Set if overflow; reset otherwise
N:	Set
C:	Set if borrow; reset otherwise

Example:

If the Accumulator contains 63H, the HL register pair contains 6000H and memory location 6000H contains 60H, the instruction

CP (HL)

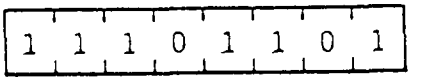
will result in the P/V flag in the F register being reset.

CPD

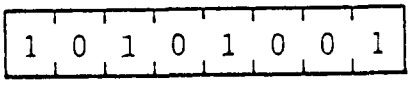
Operation: A - (HL), HL ← HL-1, BC ← BC-1

Format:

Opcode	Operands
CPD	



ED



A9

Description:

The contents of the memory location addressed by the HL register pair is compared with the contents of the Accumulator. In case of a true compare, a condition bit is set. The HL and the Byte Counter (register pair BC) are decremented.

M CYCLES: 4 T STATES: 16(4,4,3,5)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if A=(HL); reset otherwise
H:	Set if borrow from Bit 4; reset otherwise
P/V:	Set if BC-1≠0; reset otherwise
N:	Set
C:	Not Affected

Example:

If the HL register pair contains 1111H, memory location 1111H contains 3BH, the Accumulator contains 3BH, and the Byte Counter contains 0001H, then after the execution of

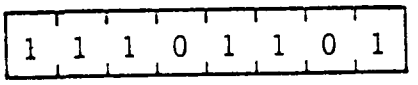
the Byte Counter will contain 0000H, the HL register pair will contain 1110H, the Z flag in the F register will be set, and the P/V flag in the F register will be reset. There will be no effect on the contents of the Accumulator or address 1111H.

CPDR

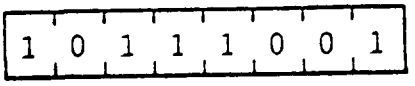
Operation: A - (HL), HL ← HL-1, BC ← BC-1

Format:

Opcode	Operands
CPDR	



ED



B9

Description:

The contents of the memory location addressed by the HL register pair is compared with the contents of the Accumulator. In case of a true compare, a condition bit is set. The HL and BC (Byte Counter) register pairs are decremented. If decrementing causes the BC to go to zero or if A = (HL), the instruction is terminated. If BC is not zero and A ≠ (HL), the program counter is decremented by 2 and the instruction is repeated. Note that if BC is set to zero prior to instruction execution, the instruction will loop through 64K bytes, if no match is found. Interrupts will be recognized and two refresh cycles will be executed after each data transfer.

For BC≠0 and A≠(HL):

M CYCLES: 5 T STATES: 21(4,4,3,5,5)

For BC=0 or A=(HL):

M CYCLES: 4 T STATES: 16(4,4,3,5)

Condition Bits Affected:

S:	Set is result is negative; reset otherwise
Z:	Set if A = (HL); reset otherwise
H:	Set if borrow from Bit 4; reset otherwise
P/V:	Set if BC-1≠0; reset otherwise
N:	Set
C:	Not affected

Example:

If the HL register pair contains 1118H, the Accumulator contains F3H, the Byte Counter contains 0007H, and memory locations have these contents:

(1118H) : 52H
(1117H) : 00H
(1116H) : F3H

then after the execution of

CPDR

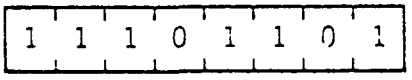
the contents of register pair HL will be 1115H, the contents of the Byte Counter will be 0004H, the P/V flag in the F register will be set, and the Z flag in the F register will be set.

CPI

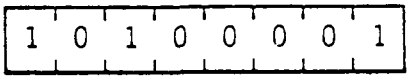
Operation: A - (HL), HL ← HL+1, BC ← BC-1

Format:

Opcode	Operands
CPI	



ED



A1

Description:

The contents of the memory location addressed by the HL register pair is compared with the contents of the Accumulator. In case of a true compare, a condition bit is set. Then HL is incremented and the Byte Counter (register pair BC) is decremented.

M CYCLES: 4 T STATES: 16(4,4,3,5)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if A=(HL); reset otherwise
H:	Set if borrow from Bit 4; reset otherwise
P/V:	Set if BC-1≠0; reset otherwise
N:	Set
C:	Not affected

Example:

If the HL register pair contains 1111H, memory location 1111H contains 3BH, the Accumulator contains 3BH, and the Byte Counter contains 0001H, then after the execution of

CPI

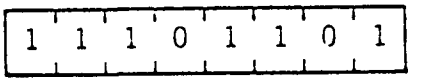
the Byte Counter will contain 0000H, the HL register pair will contain 1112H, the Z flag in the F register will be set, and the P/V flag in the F register will be reset. There will be no effect on the contents of the Accumulator or address 1111H.

CPIR

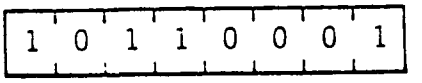
Operation: A - (HL), HL ← HL+1, BC ← BC-1

Format:

Opcode	Operands
CPIR	



ED



B1

Description:

The contents of the memory location addressed by the HL register pair is compared with the contents of the Accumulator. In case of a true compare, a condition bit is set. The HL is incremented and the Byte Counter (register pair BC) is decremented. If decrementing causes the BC to go to zero or if A = (HL), the instruction is terminated. If BC is not zero and A ≠ (HL), the program counter is decremented by 2 and the instruction is repeated. Note that if BC is set to zero before instruction execution, the instruction will loop through 64K bytes, if no match is found. Interrupts will be recognized and two refresh cycles will be executed after each data transfer.

For BC≠0 and A≠(HL):

M CYCLES: 5 T STATES: 21(4,4,3,5,5)

For BC=0 or A=(HL):

M CYCLES: 4 T STATES: 16(4,4,3,5)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if A=(HL); reset otherwise
H:	Set if borrow from Bit 4; reset otherwise
P/V:	Set if BC-1≠0; reset otherwise
N:	Set
C:	Not affected

Example:

If the HL register pair contains 1111H, the Accumulator contains F3H, the Byte Counter contains 0007H, and memory locations have these contents:

(1111H) : 52H
(1112H) : 00H
(1113H) : F3H

then after the execution of

CPIR

the contents of register pair HL will be 1114H, the contents of the Byte Counter will be 0004H, the P/V flag in the F register will be set and the Z flag in the F register will be set.

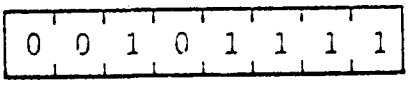
CPL

Operation: $A \leftarrow \overline{A}$

Format:

Opcode

CPL



2F

Description:

Contents of the Accumulator (register A) are inverted (1's complement).

M CYCLES: 1 T STATES: 4

Condition Bits Affected:

S:	Not affected
Z:	Not affected
H:	Set
P/V:	Not affected
N:	Set
C:	Not affected

Example:

If the contents of the Accumulator are 1011 0100, after the execution of

CPL

the Accumulator contents will be 0100 1011.

DAA

Operation: —

Format:

Opcode

DAA

0	0	1	0	0	1	1	1

27

Description:

This instruction conditionally adjusts the Accumulator for BCD addition and subtraction operations. For addition (ADD, ADC, INC) or subtraction (SUB, SBC,DEC,NEG), the following table indicates operation performed:

OPERATION	C BEFORE DAA	HEX VALUE IN UPPER DIGIT (bit 7-4)	H BEFORE DAA	HEX VALUE IN LOWER DIGIT (bit 3-0)	NUMBER ADDED TO BYTE	C AFTER DAA
ADD ADC INC	0	0-9	0	0-9	00	0
	0	0-8	0	A-F	06	0
	0	0-9	1	0-3	06	0
	0	A-F	0	0-9	60	1
	0	9-F	0	A-F	66	1
	0	A-F	1	0-3	66	1
	1	0-2	0	0-9	60	1
	1	0-2	0	A-F	66	1
	1	0-3	1	0-3	66	1
SUB SBC DEC NEG	0	0-9	0	0-9	00	0
	0	0-8	1	6-F	FA	0
	1	7-F	0	0-9	A0	1
	1	6-F	1	6-F	9A	1

M CYCLES: 1 T STATES: 4

Condition Bits Affected:

S:	Set if most significant bit of Acc. is 1 after operation; reset otherwise
Z:	Set if Acc. is zero after operation; reset otherwise
H:	See instruction
P/V:	Set if Acc. is even parity after operation; reset otherwise
N:	Not affected
C:	See instruction

Example:

If an addition operation is performed between 15 (BCD) and 27 (BCD), simple decimal arithmetic gives this result:

```
 15
+27
 42
```

But when the binary representations are added in the Accumulator according to standard binary arithmetic,

```
 0001  0101
+0010  0111
 0011  1100   3C
```

the sum is ambiguous. The DAA instruction adjusts this result so that the correct BCD representation is obtained:

```
 0011  1100
+0000  0110
 0100  0010 = 42
```

DEC IX

Operation: IX ← IX -1

Format:

Opcode	Operands	
DEC	IX	
1 1 1 1 1 1 0 1		DD
0 0 1 0 1 0 1 1		2B

Description:

The contents of the Index Register IX are decremented.

M CYCLES: 2 T STATES: 10 (4,6)

Condition Bits Affected: None

Example:

If the contents of the Index Register IX are 7649H, after the execution of

```
DEC  IX
```

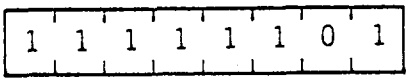
the contents of Index Register IX will be 7648H.

DEC IY

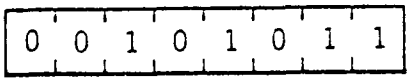
Operation: IY ← IY -1

Format:

Opcode	Operands
DEC	IY



FD



2B

Description:

The contents of the Index Register IY are decremented.

M CYCLES: 2 T STATES: 10 (4,6)

Condition Bits Affected: None

Example:

If the contents of the Index Register IY are 7649H, after the execution of

```
DEC IY
```

the contents of Index Register IY will be 7648H.

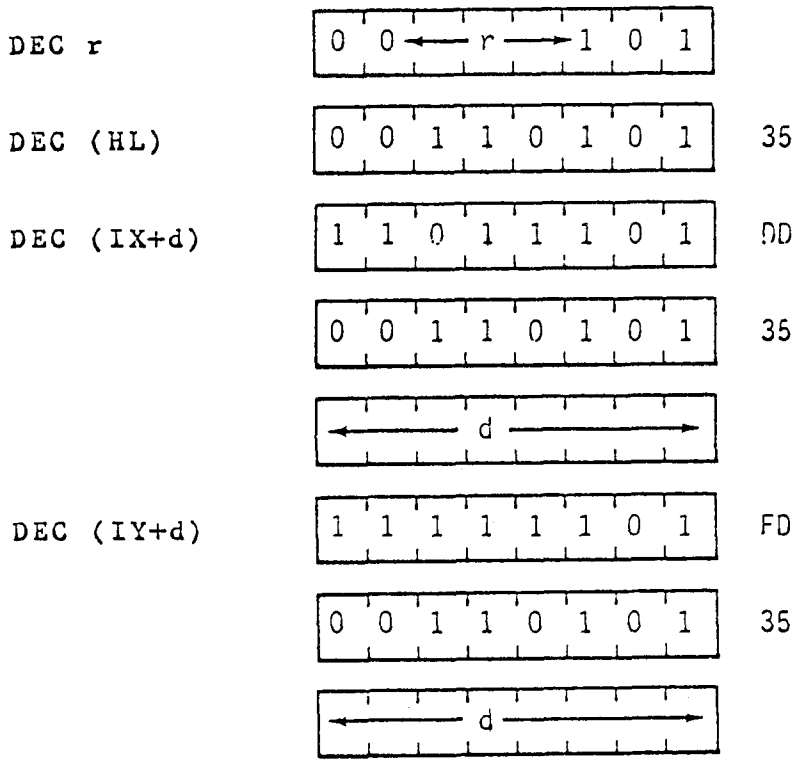
DEC m

Operation: m ← m-1

Format:

Opcode	Operands
DEC	m

The m operand is any of r, (HL),(IX+d) or (IY+d), as defined for the analogous INC instructions. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B,C,D,E,H,L or A assembled as follows in the object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

The byte specified by the m operand is decremented.

INSTRUCTION	M CYCLES	T STATES
DEC r	1	4
DEC (HL)	3	11(4,4,3)
DEC (IX+d)	6	23(4,4,3,5,4,3)
DEC (IY+d)	6	23(4,4,3,5,4,3)

Conditions Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if borrow from Bit 4, reset otherwise
P/V:	Set if m was 80H before operation; reset otherwise
H:	Set
C:	Not affected

Example:

If the D register contains byte 2AH, after the executio of

```
DEC D
```

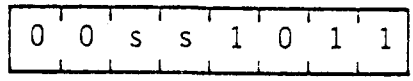
register D will contain 29H.

DEC ss

Operation: ss ← ss -1

Format:

Opcode	Operands
DEC	ss



Description:

The contents of register pair ss (any of the register pairs BC,DE,HL or SP) are decremented. Operand ss is specified as follows in the assembled object code.

Pair	ss
BC	00
DE	01
HL	10
SP	11

M CYCLES: 1 T STATES: 6

Condition Bits Affected: None

Example:

If register pair HL contains 1001H, after the execution of

```
DEC HL
```

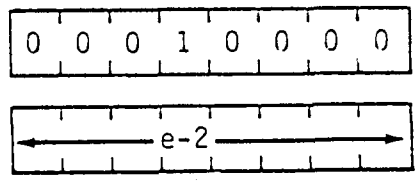
the contents of HL will be 1000H.

DJNZ, e

Operation: —

Format:

Opcode	Operand
DJNZ	e



Description:

This instruction is similar to the conditional jump instructions except that a register value is used to determine branching. The B register is decremented and if a non zero value remains, the value of the displacement e is added to the Program Counter (PC). The next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction opcode and has a range of -126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the result of decrementing leaves B with a zero value, the next instruction to be executed is taken from the location following this instruction.

If B≠0:

M CYCLES: 3 T STATES: 13(5,3,5)

If B=0:

M CYCLES: 2 T STATES: 8(5,3)

Condition Bits Affected: None

Example:

A typical software routine is used to demonstrate the use of the DJNZ instruction. This routine moves a line from an input buffer (INBUF) to an output buffer

(OUTBUF). It moves the bytes until it finds a CR, or until it has moved 80 bytes, whichever occurs first.

```
        LD          B,80          ;Set up counter
        LD          HL,Inbuf      ;Set up pointers
        LD          DE,Outbuf

OOP:    LD          A,(HL)        ;Get next byte from
                                  ;input buffer
        LD          (DE),A        ;Store in output buffer
        CP          00H           ;Is it a CR?
        JR          Z,DONE        ;Yes finished
        INC         HL            ;Increment pointers
        INC         DE
        DJNZ        LOOP          ;Loop back if 80
                                  ;bytes have not
                                  ;been moved
ONE:
```

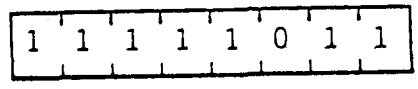
EI

Operation: IFF ← 1

Format:

Opcode

EI



FB

Description:

EI enables the maskable interrupt by setting the interrupt enable flip-flops(IFF1 and IFF2). Note that this instruction disables the maskable interrupt during its execution.

M CYCLES: 1 T STATES: 4

Condition Bits Affected: None

Example:

When the CPU executes instruction

EI

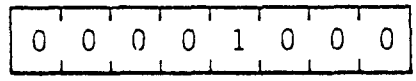
the maskable interrupt is enabled. The CPU will now respond to an Interrupt Request (INT) signal.

EX AF, AF′

Operation: AF ↔ AF′

Format:

Opcode	Operands
EX	AF,AF′



08

Description:

The two-byte contents of the register pairs AF and AF′ are exchanged. (Note: register pair AF′ consists of registers A′ and F′.)

M CYCLES: 1 T STATES: 4

Condition Bits Affected: None

Example:

If the content of register pair AF is number 9900H, and the content of register pair AF′ is number 5944H, after the instruction

EX AF,AF′

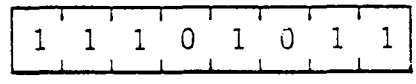
the contents of AF will be 5944H, and the contents of AF′ will be 9900H.

EX DE, HL

Operation: DE ↔ HL

Format:

Opcode	Operands
EX	DE,HL



EB

Description:

The two-byte contents of register pairs DE and HL are exchanged.

M CYCLES: 1 T STATES: 4

Condition Bits Affected: None

Example:

If the content of register pair DE is the number 2822H, and the content of the register pair HL is number 499AH, after the instruction

EX DE,HL

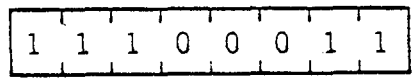
the content of register pair DE will be 499AH and the content of register pair HL will be 2822H.

EX (SP), HL

Operation: H ↔ (SP+1), L ↔ (SP)

Format:

Opcode	Operands
EX	(SP),HL



E3

Description:

The low order byte contained in register pair HL is exchanged with the contents of the memory address specified by the contents of register pair SP (Stack Pointer), and the high order byte of HL is exchanged with the next highest memory address (SP+1).

M CYCLES: 5 T STATES: 19(4,3,4,3,5

Condition Bits Affected: None

Example:

If the HL register pair contains 7012H, the SP register pair contains 8856H, the memory location 8856H contains the byte 11H, and the memory location 8857H contains the byte 22H, then the instruction

EX (SP),HL

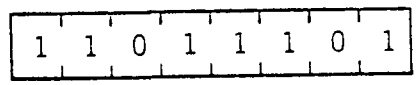
will result in the HL register pair containing number 2211H, memory location 8856H containing the byte 12H, the memory location 8857H containing the byte 70H and the Stack Pointer containing 8856H.

EX (SP), IX

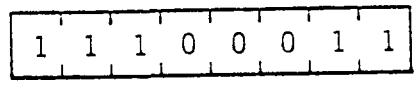
Operation: $IX_H \leftrightarrow (SP+1)$, $IX_L \leftrightarrow (SP)$

Format:

Opcode	Operands
EX	(SP),IX



DD



E3

Description:

The low order byte in Index Register IX is exchanged with the contents of the memory address specified by the contents of register pair SP (Stack Pointer), and the high order byte of IX is exchanged with the next highest memory address (SP+1).

M CYCLES: 6 T STATES: 23(4,4,3,4,3,5)

Condition Bits Affected: None

Example:

If the Index Register IX contains 3988H, the SP register pair contains 0100H, the memory location 0100H contains the byte 90H, and memory location 0101H contains byte 48H, then the instruction

EX (SP),IX

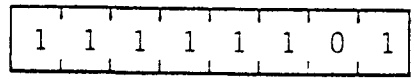
will result in the IX register pair containing number 4890H, memory location 0100H containing 88H, memory location 0101H containing 39H and the Stack Pointer containing 0100H.

EX (SP), IY

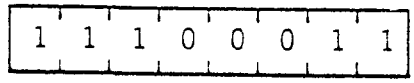
Operation: $IY_H \leftrightarrow (SP+1)$, $IY_L \leftrightarrow (SP)$

Format:

Opcode	Operands
EX	(SP),IY



FD



E3

Description:

The low order byte in Index Register IY is exchanged with the contents of the memory address specified by the contents of register pair SP (Stack Pointer), and the high order byte of IY is exchanged with the next highest memory address (SP+1).

M CYCLES: 6 T STATES: 23(4,4,3,4,3,5)

Condition Bits Affected: None

Example:

If the Index Register IY contains 3988H, the SP register pair contains 0100H, the memory location 0100H contains the byte 90H, and memory location 0101H contains byte 48H, then the instruction

```
EX (SP),IY
```

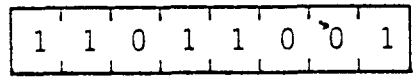
will result in the IY register pair containing number 4890H, memory location 0100H containing 88H, memory location 0101H containing 39H, and the Stack Pointer containing 0100H.

EXX

Operation: (BC) ↔ (BC'), (DE) ↔ (DE'), (HL) ↔ (HL')

Format:

Opcode	Operands
EXX	



D9

Description:

Each two-byte value in register pairs BC, DE, and HL is exchanged with the two-byte value in BC', DE', and HL', respectively.

M CYCLES: 1 T STATES: 4

Condition Bits Affected: None

Example:

If the contents of register pairs BC, DE, and HL are the numbers 445AH, 3DA2H, and 8859H, respectively, and the contents of register pairs BC', DE', and HL' are 0988H, 9300H, and 00E7H, respectively, after the instruction

EXX

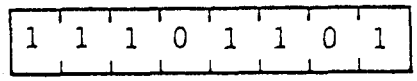
the contents of the register pairs will be as follows: BC: 0988H; DE: 9300H; HL: 00E7H; BC': 445AH; DE': 3DA2H; and HL': 8859H.

IM O

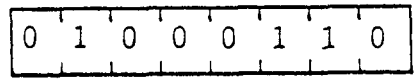
Operation: —

Format:

Opcode	Operands
IM	0



ED



46

Description:

The IM 0 instruction sets interrupt mode 0. In this mode the interrupting device can insert any instruction on the data bus and allow the CPU to execute it.

M CYCLES: 2 T STATES: 8(4,4)

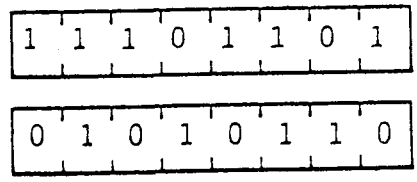
Condition Bits Affected: None

IM 1

Operation: —

Format:

Opcode	Operands
IM	1



Description:

The IM instruction sets interrupt mode 1. In this mode the processor will respond to an interrupt by executing a restart to location 0038H.

M CYCLES: 2 T STATES: 8(4,4)

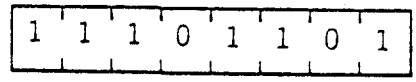
Condition Bits Affected: None

IM 2

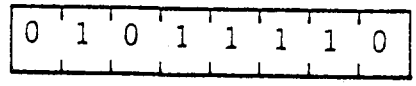
Operation: —

Format:

Opcode	Operands
IM	2



ED



5E

Description:

The IM 2 instruction sets interrupt mode 2. This mode allows an indirect call to any location in memory. With this mode the CPU forms a 16-bit memory address. The upper eight bits are the contents of the Interrupt Vector Register I and the lower eight bits are supplied by the interrupting device.

M CYCLES: 2 T STATES: 8(4,4)

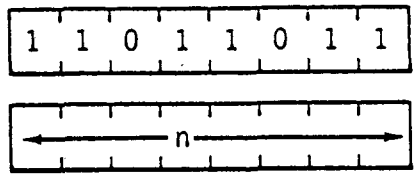
Condition Bits Affected: None

IN A, (n)

Operation: A ← (n)

Format:

Opcode	Operands
IN	A,(n)



Description:

The operand n is placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. The contents of the Accumulator also appear on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written into the Accumulator (register A) in the CPU.

M CYCLES: 3 T STATES: 11(4,3,4)

Condition Bits Affected: None

Example:

If the contents of the Accumulator are 23H and the byte 7BH is available at the peripheral device mapped to I/O port address 01H, then after the execution of

```
IN A,(01H)
```

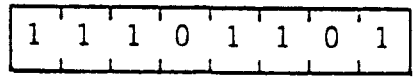
the Accumulator will contain 7BH.

IN r, (C)

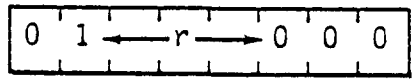
Operation: r ← (C)

Format:

Opcode	Operands
IN	r,(C)



ED



Description:

The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. The contents of Register B are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written into register r in the CPU. Register r identifies any of the CPU registers shown in the following table, which also shows the corresponding 3-bit "r" field for each. The flags will be affected, checking the input data.

Reg.	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

M CYCLES: 3 T STATES: 12(4,4,4)

Condition Bits Affected:

S:	Set if input data is negative; reset otherwise
Z:	Set if input data is zero; reset otherwise
H:	Reset
P/V:	Set if parity is even; reset otherwise
N:	Reset
C:	Not affected

Example:

If the contents of register C are 07H, the contents of register B are 10H, and the byte 7BH is available at the peripheral device mapped to I/O port address 07H, then after the execution of

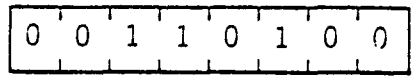
```
IN D,(C)
```

INC (HL)

Operation: (HL) ← (HL)+1

Format:

Opcode	Operands
INC	(HL)



34

Description:

The byte contained in the address specified by the contents of the HL register pair is incremented.

M CYCLES: 3 T STATES: 11(4,4,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if carry from Bit 3; reset otherwise
P/V:	Set if (HL) was 7FH before operation; reset otherwise
N:	Reset
C:	Not Affected

Example:

If the contents of the HL register pair are 3434H, and the contents of address 3434H are 82H, after the execution of

INC (HL)

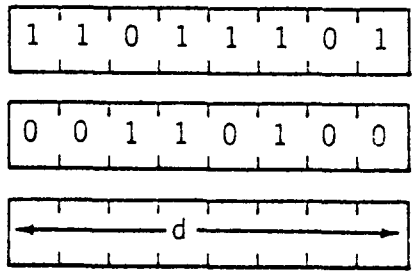
memory location 3434H will contain 83H.

INC (IX+d)

Operation: (IX+d) ← (IX+d)+1

Format:

Opcode	Operands
INC	(IX+d)



Description:

The contents of the Index Register IX (register pair IX) are added to a two's complement displacement integer d to point to an address in memory. The contents of this address are then incremented.

M CYCLES: 6 T STATES: 23(4,4,3,5,4,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if carry from Bit 3; reset otherwise
P/V:	Set if (IX+d) was 7FH before operation; reset otherwise
N:	Reset
C:	Not affected

Example:

If the contents of the Index Register pair IX are 2020H, and the memory location 2030H contains byte 34H, after the execution of

```
INC (IX+10H)
```

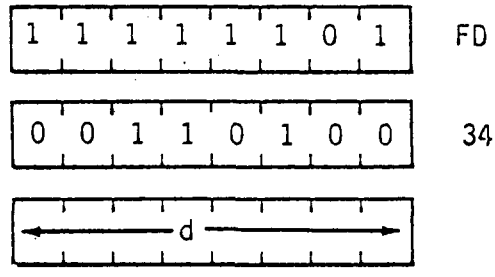
the contents of memory location 2030H will be 35H.

INC (IY+d)

Operation: (IY+d) ← (IY+d)+1

Format:

Opcode	Operands
INC	(IY+d)



Description:

The contents of the Index Register IY (register pair IY) are added to a two's complement displacement integer d to point to an address in memory. The contents of this address are then incremented.

M CYCLES: 6 T STATES: 23(4,4,3,5,4,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if carry from Bit 3; reset otherwise
P/V:	Set if (IY+d) was 7FH before operation; reset otherwise
N:	Reset
C:	Not Affected

Example:

If the contents of the Index Register pair IY are 2020H, and the memory location 2030H contain byte 34H, after the execution of

```
INC (IY+10H)
```

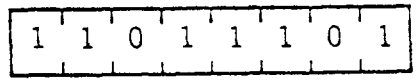
the contents of memory location 2030H will be 35H.

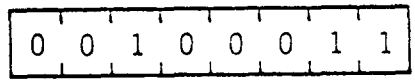
INC IX

Operation: IX ← IX + 1

Format:

Opcode	Operands
INC	IX





Description:

The contents of the Index Register IX are incremented.

M CYCLES: 2 T STATES: 10(4,6)

Condition Bits Affected: None

Example:

If the Index Register IX contains the integer 3300H after the execution of

```
INC IX
```

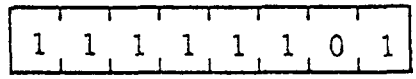
the contents of Index Register IX will be 3301H.

INC IY

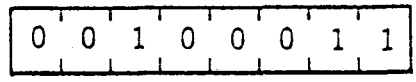
Operation: IY ← IY + 1

Format:

Opcode	Operands
INC	IY



FD



23

Description:

The contents of the Index Register IY are incremented.

M CYCLES: 2 T STATES: 10(4,6)

Condition Bits Affected: None

Example:

If the contents of the Index Register are 2977H, after the execution of

```
INC IY
```

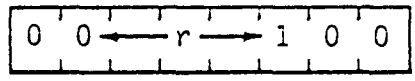
the contents of Index Register IY will be 2978H.

INC r

Operation: r←r+1

Format:

Opcode	Operands
INC	r



Description:

Register r is incremented. r identifies any of the registers A,B, C,D,E,H or L, assembled as follows in the object code.

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M CYCLES: 1 T STATES: 4

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if carry from Bit 3; reset otherwise
P/V:	Set if r was 7FH before operation; reset otherwise
N:	Reset
C:	Not affected

Example:

If the contents of register D are 28H, after the execution of

```
INC D
```

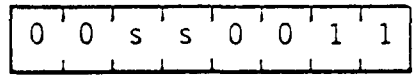
the contents of register D will be 29h.

INC ss

Operation: ss ← ss + 1

Format:

Opcodes	Operands
INC	ss



Description:

The contents of register pair ss (any of register pairs BC, DE,HL or SP) are incremented. Operand ss is specified as follows in the assembled object code.

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

M CYCLES: 1 T STATES: 6

Condition Bits Affected: None

Example:

If the register pair contains 1000H, after the execution of

```
INC HL
```

HL will contain 1001H.

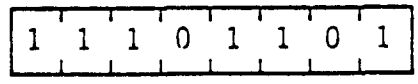
IND

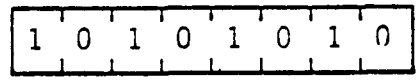
Operation: (HL)←(C), B←B-1, HL←HL-1

Format:

Opcode

IND





Description:

The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its contents are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the CPU. The contents of the HL register pair are placed on the address bus and the input byte is written into the corresponding location of memory. Finally the byte counter and register pair HL are decremented.

M CYCLES: 4 T STATES: 16(4,5,3,4)

Condition Bits Affected:

S: Unknown
Z: Set if B-1=0; reset otherwise
H: Unknown
P/V: Unknown
N: Set
C: unknown

Example:

If the contents of register C are 07H, the contents of register B are 10H, the contents of the HL register pair are 1000H, and the byte 7BH is available at the

peripheral device mapped to I/O port address 07H, then after the execution of

```
IND
```

memory location 1000H will contain 7BH, the HL register pair will contain 0FFFH, and register B will contain 0FH.

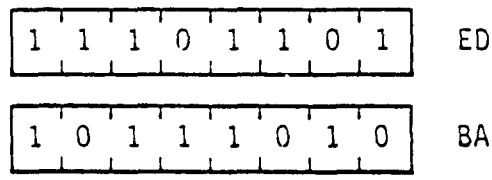
INDR

Operation: (HL) ← (C), B ← B-1, HL ← HL-1

Format:

Opcode

INDR



Description:

The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B is used as a byte counter, and its contents are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the HL register pair are placed on the address bus and the input byte is written into the corresponding location of memory. Then HL and the byte counter are decremented. If decrementing causes B to go to zero, the instruction is terminated. If B is not zero, the PC is decremented by two and the instruction repeated. Note that if B is set to zero prior to instruction execution, 256 bytes of data will be input. Interrupts will be recognized and two refresh cycles will be executed after each data transfer.

If B≠0:

M CYCLES: 5 T STATES: 21(4,5,3,4,5)

If B=0:

M CYCLES: 4 T STATES: 16(4,5,3,4)

Condition Bits Affected:

S:	Unknown
Z:	Set
H:	Unknown
P/V:	Unknown
N:	Set
C:	unknown

Example:

If the contents of register C are 07H, the contents of register B are 03H, the contents of the HL register pair are 1000H, and the following sequence of bytes are available at the peripheral device mapped to I/O port address 07H:

51H
A9H
03H

then after the execution of

INDR

the HL register pair will contain 0FFDH, register B will contain zero, and memory locations will have contents as follows:

Location	Contents
0FFEH	03H
0FFFH	A9H
1000H	51H

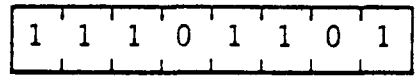
INI

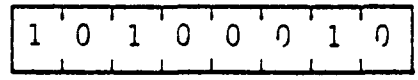
Operation: (HL) ← (C) , B ← B-1 , HL ← HL + 1

Format:

Opcode

INI





Description:

The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its contents are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the CPU. The contents of the HL register pair are then placed on the address bus and the input byte is written into the corresponding location of memory. Finally the byte counter is decremented and register pair HL is incremented.

M CYCLES: 4 T STATES: 16(4,5,3,4)

Condition Bits Affected:

S: Unknown
Z: Set if B-1=0; reset otherwise
H: Unknown
P/V: Unknown
N: Set
C: unknown

Example:

If the contents of register C are 07H, the contents of register B are 10H, the contents of the HL register pair are 1000H, and the byte 7BH is available at the peripheral device mapped to I/O port address 07H, then

after the execution of

INI

memory location 1000H will contain 7BH, the HL register pair will contain 1001H, and register B will contain 0FH.

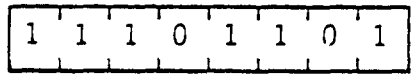
INIR

Operation: (HL) ← (C) , B ← B-1 , HL ← HL + 1

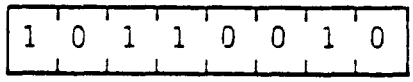
Format:

Opcode

INIR



ED



B2

Description:

The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B is used top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the CPU. The contents of the HL register pair are placed on the address bus and the input byte is written into the corresponding location of memory. Then register pair HL is incremented, the byte counter is decremented. If decrementing causes B to go to zero, the instruction is terminated. If B is not zero, the PC is decremented by two and the instruction repeated. Note that if B is set to zero prior to instruction execution, 256 bytes of data will be input. Interrupts will be recognized and two refresh cycles will be executed after each data transfer.

If B≠0:

M CYCLES: 5 T STATES: 21(4,5,3,4,5)

If B=0:

M CYCLES: 4 T STATES: 16(4,5,3,4)

Condition Bits Affected:

S: Unknown
Z: Set
H: Unknown
P/V: Unknown
N: Set
C: unknown

Example:

If the contents of register C are 07H, the contents of register B are 03H, the contents of the HL register pair are 1000H, and the following sequence of bytes are available at the peripheral device mapped to I/O port of address 07H:

51H
A9H
03H

then after the execution of

INIR

the HL register pair will contain 1003H, register B will contain zero, and memory locations will have contents as follows:

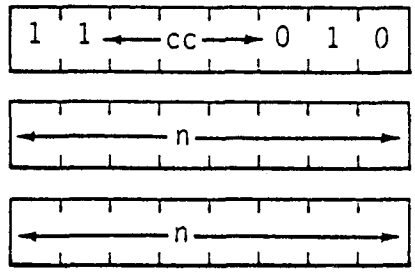
Location	Contents
1000H	51H
1001H	A9H
1002H	03H

JP cc, nn

Operation: IF cc TRUE, PC ← nn

Format:

Opcode	Operands
JP	cc,nn



Note: The first n operand in this assembled object code is the low order byte of a 2-byte memory address.

Description:

If condition cc is true, the instruction loads operand nn into register pair PC (Program Counter), and the program continues with the instruction beginning at address nn. If condition cc is false, the Program Counter is incremented as usual, and the program continues with the next sequential instruction. Condition cc is programmed as one of eight status which corresponds to condition bits in the Flag Register (register F). These eight status are defined in the table below which also specifies the corresponding cc bit fields in the assembled object code.

cc	CONDITION	RELEVANT FLAG
000	NZ non zero	Z
001	Z zero	Z
010	NC no carry	C
011	C carry	C
100	PO parity odd	P/V
101	PE parity even	P/V
110	P sign positive	S
111	M sign negative	S

M CYCLES: 3 T STATES: 10(4,3,3)

Condition Bits Affected: None

Example:

If the Carry Flag (C flag in the F register) is set and the contents of address 1520 are 03H, after the execution of

```
JP C,1520H
```

the Program Counter will contain 1520H, and on the next machine cycle the CPU will fetch from address 1520H the byte 03H.

JP (HL)

Operation: PC ← HL

Format:

Opcode	Operands
JP	(HL)

1	1	1	0	1	0	0	1	E9

Description:

The Program Counter (register pair PC) is loaded with the contents of the HL register pair. The next instruction is fetched from the location designated by the new contents of the PC.

M CYCLES: 1 T STATES: 4

Condition Bits Affected: None

Example:

If the contents of the Program Counter are 1000H and the contents of the HL register pair are 4800H, after the execution of

```
JP (HL)
```

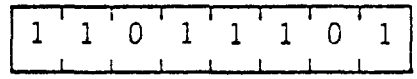
the contents of the Program Counter will be 4800H.

JP (IX)

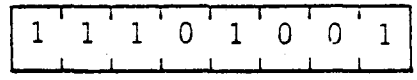
Operation: PC ← IX

Format:

Opcode	Operands
JP	(IX)



DD



E9

Description:

The Program Counter (register pair PC) is loaded with the contents of the IX Register Pair (Index Register IX). The next instruction is fetched from the location designated by the new contents of the PC.

M CYCLES: 2 T STATES: 8(4,4)

Condition Bits Affected: None

Example:

If the contents of the Program Counter are 1000H, and the contents of the IX Register Pair are 4800H, after the execution of

```
JP (IX)
```

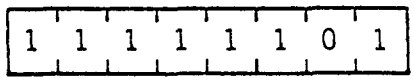
the contents of the Program Counter will be 4800H.

JP (IY)

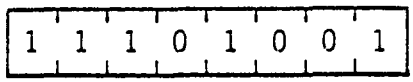
Operation: PC ← IY

Format:

Opcode	Operands
JP	(IY)



FD



E9

Description:

The Program Counter (register pair PC) is loaded with the contents of the IY register pair (Index Register IY). The next instruction is fetched from the location designated by the new contents of the PC.

M CYCLES: 2 T STATES: 8(4,4)

Condition Bits Affected: None

Example:

If the contents of the Program Counter are 1000H and the contents of the IY Register Pair are 4800H, after the execution of

JP (IY)

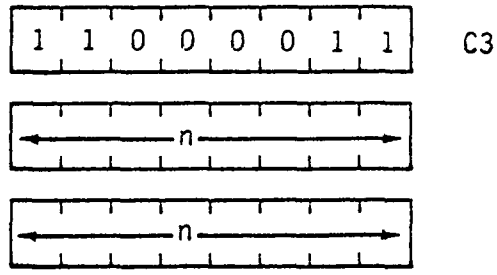
the contents of the Program Counter will be 4800H.

JP nn

Operation: PC ← nn

Format:

Opcode	Operands
JP	nn



Note: The first operand in this assembled object code is the low order byte of a 2-byte address.

Description:

Operand nn is loaded into register pair PC (Program Counter) and points to the address of the next program instruction to be executed.

M CYCLES: 3 T STATES: 10(4,3,3)

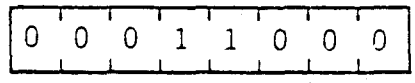
Condition Bits Affected: None

JR e

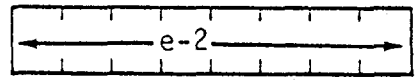
Operation: PC ← PC + e

Format:

Opcode	Operand
JR	e



18



Description:

This instruction provides for unconditional branching to other segments of a program. The value of the displacement e is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. This jump is measured from the address of the instruction opcode and has a range of -126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

M CYCLES: 3 T STATES: 12(4,3,5)

Condition Bits Affected: None

Example:

To jump forward 5 locations from address 480, the following assembly language statement is used:

```
JR $+5
```

The resulting object code and final PC value is shown below:

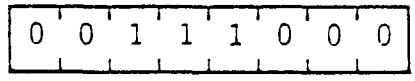
Location	Instruction	
480	18	
481	03	
482	—	
483	—	
484	—	
485	←	PC after jump

JR C, e

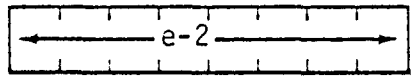
Operation: If C = 0, continue
If C = 1, PC ← PC + e

Format:

Opcode	Operands
JR	C,e



38



Description:

This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Carry Flag. If the flag is equal to a '1', the value of the displacement e is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction opcode and has a range of -126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the flag is equal to a '0', the next instruction to be executed is taken from the location following this instruction.

If condition is met:

M CYCLES: 3 T STATES: 12(4,3,5)

If condition is not met:

M CYCLES: 2 T STATES: 7(4,3)

Condition Bits Affected: None

Example:

The Carry Flag is set and it is required to jump back 4 locations from 480. The assembly language statement is:

```
JR C,$-4
```

The resulting object code and final PC value is shown below:

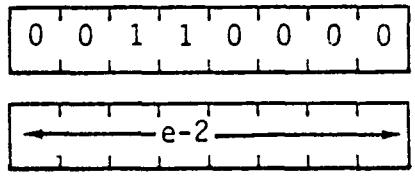
Location	Instruction
47C	← PC after jump
47D	—
47E	—
47F	—
480	38
481	FA (2's complement-6)

JR NC, e

Operation: If C = 1, continue
If C = 0, PC ← PC + e

Format:

Opcode	Operands
JR	NC,e



Description:

This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Carry Flag. If the flag is equal to '0', the value of the displacement e is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction opcode and has a range of -126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the flag is equal to a '1', the next instruction to be executed is taken from the location following this instruction.

If the condition is met:

M CYCLES: 3 T STATES: 12(4,3,5)

If the condition is not met:

M CYCLES: 7 T STATES: 7(4,3)

Condition Bits Affected: None

Example:

The Carry Flag is reset and it is required to repeat the jump instruction. The assembly language statement is:

```
JR NC,$
```

The resulting object code and PC after the jump are shown below:

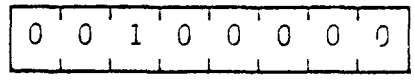
Location	Instruction
480	30 ← PC after jump
481	00

JR NZ, e

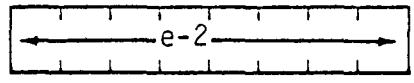
Operation: If Z = 1, continue
If Z = 0, PC ← PC + e

Format:

Opcode	Operands
JR	NZ,e



20



Description:

This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Zero Flag. If the flag is equal to a '0', the value of the displacement e is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction opcode and has a range of -126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the Zero Flag is equal to a '1', the next instruction to be executed is taken from the location following this instruction.

If the condition is met:

M CYCLES: 3 T STATES: 12(4,3,5)

If the condition is not met:

M CYCLES: 2 T STATES: 7(4,3)

Condition Bits Affected: None

Example:

The Zero Flag is reset and it is required to jump back 4 locations from 480. The assembly language statement is:

```
JR NZ,$-4
```

The resulting object code and final PC value is shown below:

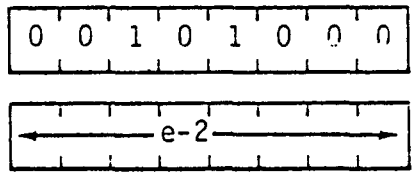
Location	Instruction	
47C	←	PC after jump
47D	—	
47E	—	
47F	—	
480	20	
481	FA	(2' complement-6)

JR Z, e

Operation: If Z = 0, continue
If Z = 1, PC ← PC + e

Format:

Opcode	Operands
JR	Z,e



Description:

This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Zero Flag. If the flag is equal to a '1', the value of the displacement e is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction opcode and has a range of -126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the Zero Flag is equal to a '0', the next instruction to be executed is taken from the location following this instruction.

If the condition is met:

M CYCLES: 3 T STATES: 12(4,3,5)

If the condition is not net:

M CYCLES: 2 T STATES: 7(4,3)

Condition Bits Affected: None

Example:

The Zero Flag is set and it is required to jump forward 5 locations from address 300. The following assembly language statement is used:

```
JR Z,$ +5
```

The resulting object code and final PC value is shown below:

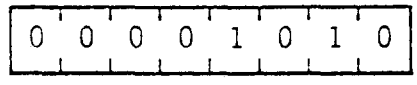
Location	Instruction	
300	28	
301	03	
302	—	
303	—	
304	—	
305	←	PC after jump

LD A, (BC)

Operation: A ← (BC)

Format:

Opcode	Operands
LD	A, (BC)



0A

Description:

The contents of the memory location specified by the contents of the BC register pair are loaded into the Accumulator.

M CYCLES: 2 T STATES: 7(4,3)

Condition Bits Affected: None

Example:

If the BC register pair contains the number 4747H, and memory address 4747H contains the byte 12H, then the instruction

LD A, (BC)

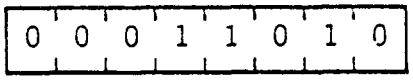
will result in byte 12H in register A.

LD A, (DE)

Operation: A ← (DE)

Format:

Opcode	Operands
LD	A, (DE)



1A

Description:

The contents of the memory location specified by the register pair DE are loaded into the Accumulator.

M CYCLES: 2 T STATES: 7(4,3)

Condition Bits Affected: None

Example:

If the DE register pair contains the number 30A2H and memory address 30A2H contains the byte 22H, then the instruction

LD A, (DE)

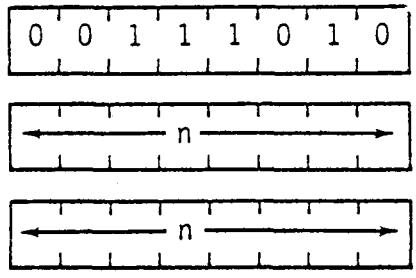
will result in byte 22H in register A.

LD A, (nn)

Operation: A ← (nn)

Format:

Opcode	Operands
LD	A, (nn)



Description:

The contents of the memory location specified by the operands nn are loaded into the Accumulator. The first n operand is the low order byte of a two-byte memory address.

M CYCLES: 4 T STATES: 13(4,3,3,3)

Condition Bits Affected: None

Example:

If the contents of nn is number 8832H, and the content of memory address 8832H is byte 04H, after the instruction

LD A, (nn)

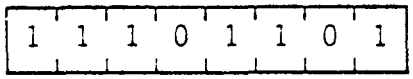
byte 04H will be in the Accumulator.

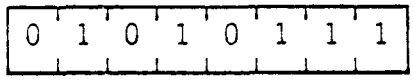
LD A, I

Operation: A ← I

Format:

Opcode	Operands
LD	A, I





Description:

The contents of the Interrupt Vector Register I are loaded into the Accumulator.

M CYCLES:2 T STATES: 9(4,5)

Condition Bits Affected:

S:	Set if I-Reg. is negative; reset otherwise
Z:	Set if I-Reg. is zero; reset otherwise
H:	Reset
P/V:	Contains contents of IFF2
N:	Reset
C:	Not affected

Example:

If the Interrupt Vector Register contains the byte 4AH, after the execution of

```
LD A, I
```

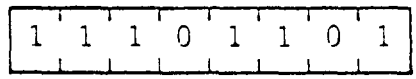
the accumulator will also contain 4AH.

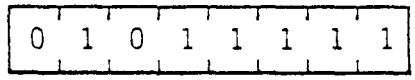
LD A, R

Operation: A ← R

Format:

Opcode	Operands
LD	A,R





Description:

The contents of Memory Refresh Register R are loaded into the Accumulator.

M CYCLES: 2 T STATES: 9(4,5)

Condition Bits Affected:

S:	Set if R-Reg. is negative; reset otherwise
Z:	Set if R-Reg. is zero; reset otherwise
H:	Reset
P/V:	Contains contents of IFF2
N:	Reset
C:	Not affected

Example:

If the Memory Refresh Register contains the byte 4AH, after the execution of

```
LD A,R
```

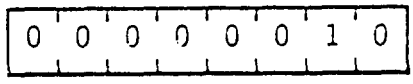
the Accumulator will also contain 4AH.

LD (BC), A

Operation: (BC) ← A

Format:

Opcode	Operands
LD	(BC),A



02

Description:

The contents of the Accumulator are loaded into the memory location specified by the contents of the register pair BC.

M CYCLES: 2 T STATES: 7(4,3)

Condition Bits Affected: None

Example:

If the Accumulator contains 7AH and the BC register pair contains 1212H the instruction

```
LD (BC),A
```

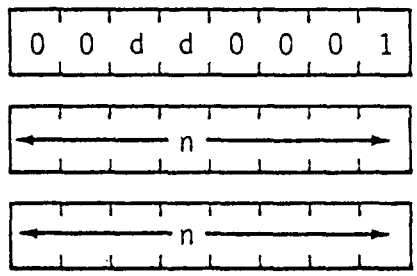
will result in 7AH being in memory location 1212H.

LD dd, nn

Operation: dd ← nn

Format:

Opcode	Operands
LD	dd, nn



Description:

The two-byte integer nn is loaded into the dd register pair, where dd defines the BC, DE, HL, or SP register pairs, assembled as follows in the object code:

Pair	dd
BC	00
DE	01
HL	10
SP	11

The first n operand in the assembled object code is the low order byte.

M CYCLES: 3 T STATES: 10(4,3,3)

Condition Bits Affected: None

Example:

After the execution of

```
LD HL, 5000H
```

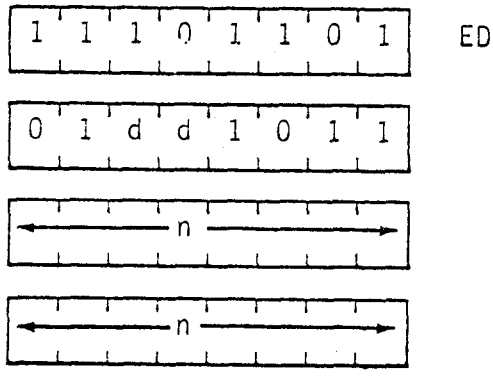
the contents of the HL register pair will be 5000H.

LD dd, (nn)

Operation: $dd_H \leftarrow (nn+1)$, $dd_L \leftarrow (nn)$

Format:

Opcode	Operands
LD	dd,(nn)



Description:

The contents of address nn are loaded into the low order portion of register pair dd, and the contents of the next highest memory address nn+1 are loaded into the high order portion of dd. Register pair dd defines BC, DE, HL, or SP register pairs, assembled as follows in the object code:

Pair	dd
BC	00
DE	01
HL	10
SP	11

The first n operand in the assembled object code above is the low order byte of (nn).

M CYCLES: 6 T STATES: 20(4,4,3,3,3,3)

Condition Bits Affected: None

Example:

If Address 2130H contains 65H and address 2131M contains 78H after the instruction

```
LD BC,(2130H)
```

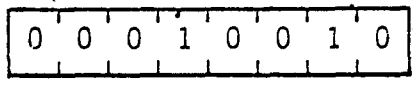
the BC register pair will contain 7865H.

LD (DE), A

Operation: (DE) ← A

Format:

Opcode	Operands
LD	(DE),A



12

Description:

The contents of the Accumulator are loaded into the memory location specified by the DE register pair.

M CYCLES: 2 T STATES: 7(4,3)

Condition Bits Affected: None

Example:

If the contents of register pair DE are 1128H, and the Accumulator contains byte A0H, the instruction

```
LD (DE),A
```

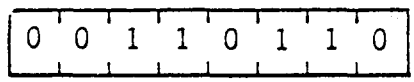
will result in A0H being in memory location 1128H.

LD (HL), n

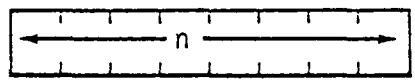
Operation: (HL) ← n

Format:

Opcode	Operand
LD	(HL),n



36



Description:

Integer n is loaded into the memory address specified by the contents of the HL register pair.

M CYCLES: 3 T STATES: 10(4,3,3)

Condition Bits Affected: None

Example:

If the HL register pair contains 4444H, the instruction

LD (HL), 28H

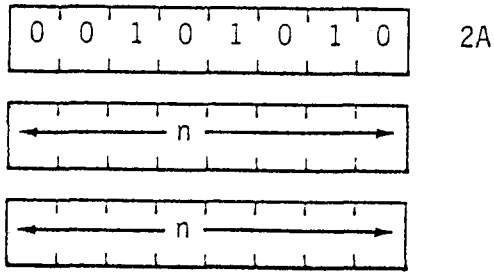
will result in the memory location 4444H containing the byte 28H.

LD HL, (nn)

Operation: H ← (nn+1), L ← (nn)

Format:

Opcode	Operands
LD	HL,(nn)



Description:

The contents of memory address nn are loaded into the low order portion of register pair HL (register L), and the contents of the next highest memory address nn+1 are loaded into the high order portion of HL (register H). The first n operand in the assembled object code above is the low order byte of nn.

M CYCLES: 5 T STATES: 16(4,3,3,3,3)

Condition Bits Affected: None

Example:

If address 4545H contains 37H and address 4546H contains A1H after the instruction

LD HL,(4545H)

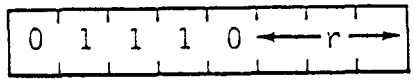
the HL register pair will contain A137H.

LD (HL), r

Operation: (HL) ← r

Format:

Opcode	Operands
LD	(HL), r



Description:

The contents of register r are loaded into the memory location specified by the contents of the HL register pair. The symbol r identifies register A, B, C, D, E, H or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M CYCLES: 2 T STATES: 7(4,3)

Condition Bits Affected: None

Example:

If the contents of register pair HL specifies memory location 2146H, and the B register contains the byte 29H, after the execution of

```
LD (HL), B
```

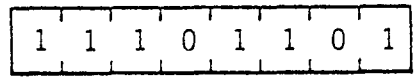
memory address 2146H will also contain 29H.

LD I, A

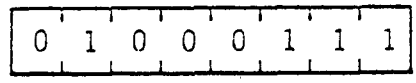
Operation: I ← A

Format:

Opcode	Operands
LD	I,A



ED



47

Description:

The contents of the Accumulator are loaded into the Interrupt Control Vector Register, I.

M CYCLES: 2 T STATES: 9(4,5)

Condition Bits Affected: None

Example:

If the Accumulator contains the number 81H, after the instruction

```
LD I,A
```

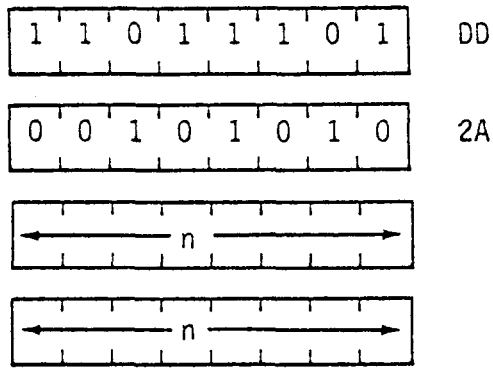
the Interrupt Vector Register will also contain 81H.

LD IX, (nn)

Operation: $IX_H \leftarrow (nn+1)$, $IX_L \leftarrow (nn)$

Format:

Opcode	Operands
LD	IX,(nn)



Description:

The contents of the address nn are loaded into the low order portion of Index Register IX, and the contents of the next highest memory address nn+1 are loaded into the high order portion of IX. The first n operand in the assembled object code above is the low order byte of nn.

M CYCLES: 6 T STATES: 20(4,4,3,3,3,3)

Condition Bits Affected: None

Example:

If address 6666H contains 92H and address 6667H contains DAH, after the instruction

```
LD IX,(6666H)
```

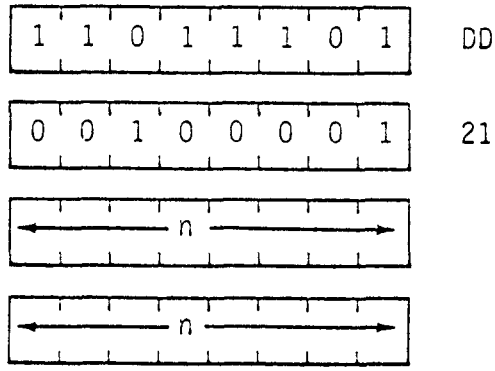
the Index Register IX will contain DA92H.

LD IX, nn

Operation: IX ← nn

Format:

Opcode	Operands
LD	IX,nn



Description:

Integer nn is loaded into the Index Register IX. The first n operand in the assembled object code above is the low order byte.

M CYCLES: 4 T STATES: 14(4,4,3,3)

Condition Bits Affected: None

Example:

After the instruction

```
LD IX,45A2H
```

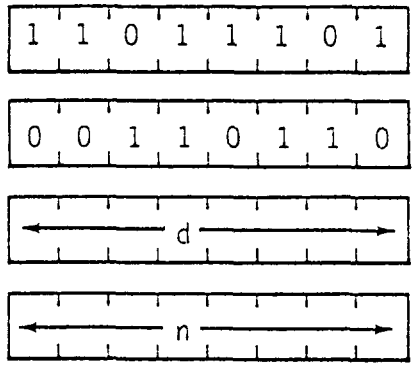
the Index Register will contain integer 45A2H.

LD (IX+d), n

Operation: (IX+d) ← n

Format:

Opcode	Operands
LD	(IX+d), n



Description:

The n operand is loaded into the memory address specified by the sum of the contents of the Index Register IX and the two's complement displacement operand d.

M CYCLES: 5 T STATES: 19(4,4,3,5,3)

Condition Bits Affected: None

Example:

If the Index Register IX contains the number 219AH the instruction

LD (IX+5H), 5AH

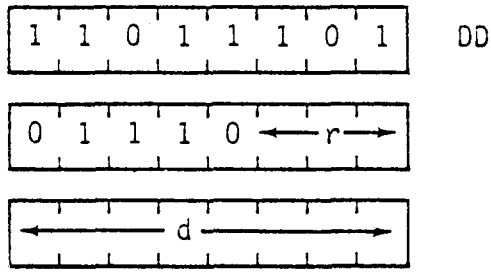
would result in the byte 5AH in the memory address 219FH.

LD (IX+d), r

Operation: (IX+d) ← r

Format:

Opcode	Operands
LD	(IX+d), r



Description:

The contents of register r are loaded into the memory address specified by the contents of Index Register IX summed with d, a two's complement displacement integer. The symbol r identifies register A, B, C, D, E, H or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M CYCLES: 5 T STATES: 19(4,4,3,5,3)

Condition Bits Affected: None

Example:

If the C register contains the byte 1CH, and the Index Register IX contains 3100H, then the instruction

```
LD (IX+6H), C
```

will perform the sum 3100H + 6H and will load 1CH into memory location 3106H.

LD IY, nn

Operation: IY ← nn

Format:

Opcode	Operands	
LD	IY,nn	
1 1 1 1 1 1 0 1		FD
0 0 1 0 0 0 0 1		21
n		
n		

Description:

Integer nn is loaded into the Index Register IY. The first n operand in the assembled object code above is the low order byte.

M CYCLES: 4 T STATES: 14(4,4,3,3)

Condition Bits Affected: None

Example:

After the instruction:

LD IY,7733H

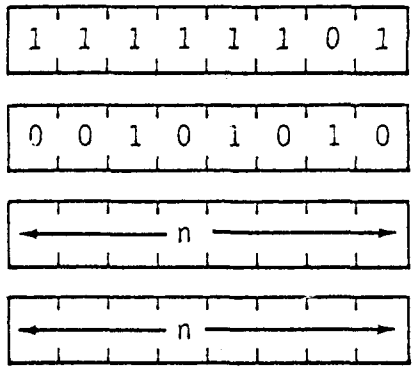
the Index Register IY will contain the integer 7733H.

LD IY, (nn)

Operation: $IY_H \leftarrow (nn+1)$, $IY_L \leftarrow (nn)$

Format:

Opcode	Operands
LD	IY,(nn)



Description:

The contents of address nn are loaded into the low order portion of Index Register IY, and the contents of the next highest memory address nn+1 are loaded into the high order portion of IY. The first n operand in the assembled object code above is the low order byte of nn.

M CYCLES: 6 T STATES: 20(4,4,3,3,3,3)

Condition Bits Affected: None

Example:

If address 6666H contains 92H and address 6667H contains DAH, after the instruction

```
LD IY,(6666H)
```

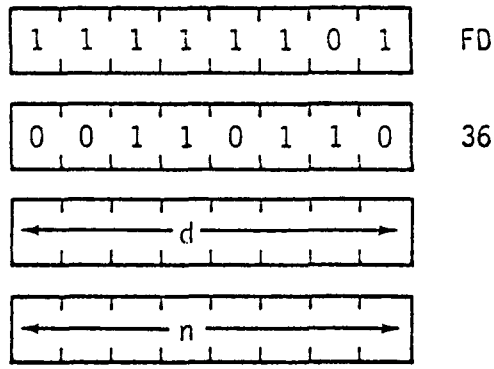
the Index Register IY will contain DA92H.

LD (IY+d), n

Operation: (IY+d) ← n

Format:

Opcode	Operands
LD	(IY+d),n



Description:

Integer n is loaded into the memory location specified by the contents of the Index Register summed with a displacement integer d.

M CYCLES: 5 T STATES: 19(4,4,3,5,3)

Condition Bits Affected: NONE

Example:

If the Index Register IY contains the number A940H, the instruction

LD (IY+10H), 97H

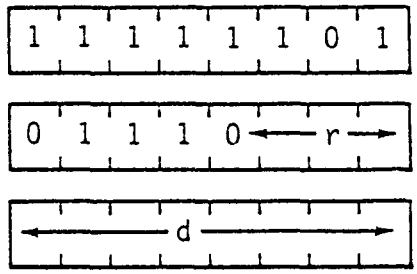
would result in byte 97 in memory location A950H.

LD (IY+d), r

Operation: (IY+d) ← r

Format:

Opcode	Operands
LD	(IY+d), r



Description:

The contents of register r are loaded into the memory address specified by the sum of the contents of the Index Register IY and d, a two's complement displacement integer. The symbol r is specified according to the following table.

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M CYCLES: 5 T STATES: 19(4,4,3,5,3)

Condition Bits Affected: None

Example:

If the C register contains the byte 48H, and the Index Register IY contains 2A11H, then the instruction

```
LD (IY+4H), C
```

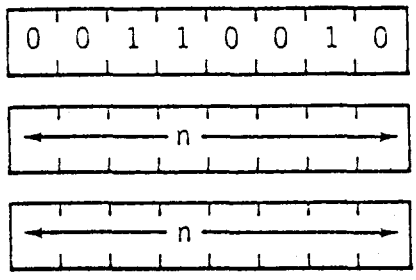
ill perform the sum 2A11H + 4H, and will load 48H into memory location 2A15.

LD (nn), A

Operation: (nn) ← A

Format:

Opcode	Operands
LD	(nn),A



Description:

The contents of the Accumulator are loaded into the memory address specified by the operands nn. The first n operand in the assembled object code above is the low order byte of nn.

M CYCLES: 4 T STATES: 13(4,3,3,3)

Condition Bits Affected: None

Example:

If the contents of the Accumulator are byte D7H, after the execution of

```
LD (3141H),A
```

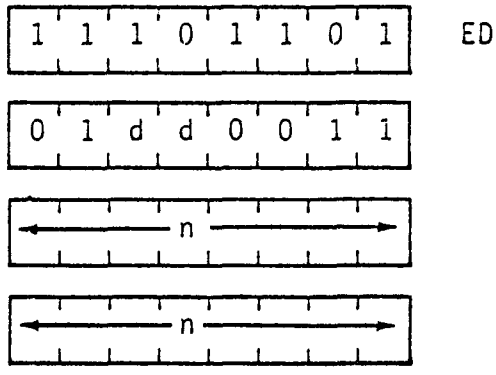
D7H will be in memory location 3141H.

LD (nn), dd

Operation: $(nn+1) \leftarrow dd_H$, $(nn) \leftarrow dd_L$

Format:

Opcode	Operands
LD	(nn),dd



Description:

The low order byte of register pair dd is loaded into memory address nn ; the upper byte is loaded into memory address nn+1 . Register pair dd defines either BC, DE, HL, or SP, assembled as follows in the object code:

Pair	dd
BC	00
DE	01
HL	10
SP	11

The first n operand in the assembled object code is the low order byte of a two byte memory address.

M CYCLES: 6 T STATES: 20(4,4,3,3,3,3)

Condition Bits Affected: None

Example:

If register pair BC contains the number 4644H, the instruction

```
LD (1000H),BC
```

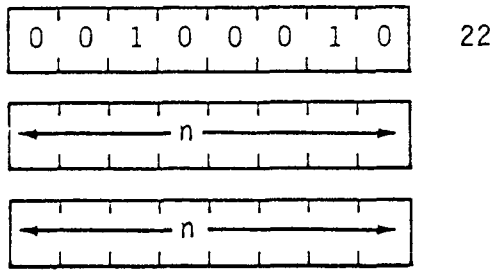
will result in 44H in memory location 1000H, and 46H in memory location 1001H.

LD (nn), HL

Operation: (nn+1) ← H , (nn) ← L

Format:

Opcode	Operands
LD	(nn) ,HL



Description.

The contents of the low order portion of register pair HL (register L) are loaded into memory address nn , and the contents of the high order portion of HL (register H) are loaded into the next highest memory address nn+1 . The first n operand in the assembled object code above is the low order byte of nn.

M CYCLES: 5 T STATES: 16(4,3,3,3,3)

Condition Bits Affected: None

Example:

If the content of register pair HL is 483AH, after the instruction

LD (B229H),HL

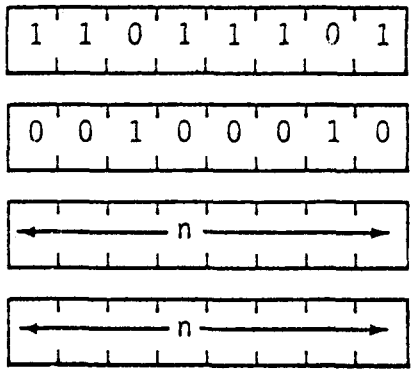
address B229H) will contain 3AH, and address B22AH will contain 48H.

LD (nn), IX

Operation: $(nn+1) \leftarrow IX_H$, $(nn) \leftarrow IX_L$

Format:

Opcode	Operands
LD	(nn),IX



Description:

The low order byte in Index Register IX is loaded into memory address nn ; the upper order byte is loaded into the next highest address nn+1 . The first n operand in the assembled object code above is the low order byte of nn.

M CYCLES: 6 T STATES: 20(4,4,3.3,3,3)

Condition Bits Affected: None

Example:

If the Index Register IX contains 5A30H, after the instruction

LD (4392H),IX

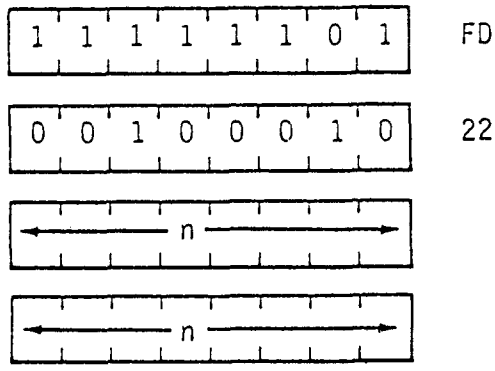
memory location 4392H will contain number 30H and location 4393H will contain 5AH.

LD (nn), IY

Operation: $(nn+1) \leftarrow IY_H$, $(nn) \leftarrow IY_L$

Format:

Opcode	Operands
LD	(nn),IY



Description:

The low order byte in Index Register IY is loaded into memory address nn ; the upper order byte is loaded into memory location nn+1 . The first n operand in the assembled object code above is the low order byte of nn.

M CYCLES: 6 T STATES: 20(4,4,3,3,3,3)

Condition Bits Affected: None

Example:

If the Index Register IY contains 4174H after the instruction

LD 8838H,IY

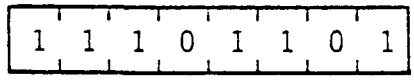
memory location 8838H will contain number 74H and memory location 8839H will contain 41H.

LD R, A

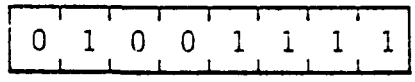
Operation: R ← A

Format:

Opcode	Operands
LD	R,A



ED



4F

Description:

The contents of the Accumulator are loaded into the Memory Refresh register R.

M CYCLES: 2 T STATES: 9(4,5)

Condition Bits Affected: None

Example:

If the Accumulator contains the number B4H, after the instruction

LD R,A

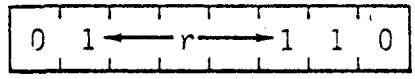
the Memory Refresh Register will also contain B4H.

LD r, (HL)

Operation: r ← (HL)

Format:

Opcode	Operands
LD	r, (HL)



Description:

The eight-bit contents of memory location (HL) are loaded into register r, where r identifies register A, B, C, D, E, H or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M CYCLES: 2 T STATES: 7(4,3)

Condition Bits Affected: None

Example:

If register pair HL contains the number 75A1H, and memory address 75A1H contains the byte 58H, the execution of

```
LD C, (HL)
```

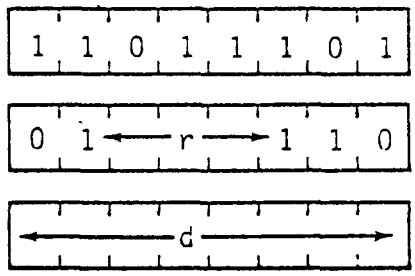
will result in 58H in register C.

LD r, (IX+d)

Operation: r ← (IX+d)

Format:

Opcode	Operands
LD	r, (IX+d)



Description:

The operand (IX+d) (the contents of the Index Register IX summed with a displacement integer d) is loaded into register r, where r identifies register A, B, C, D, E, H or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M CYCLES: 5 T STATES: 19(4,4,3,5,3)

Condition Bits Affected: None

Example:

If the Index Register IX contains the number 25AFH, the instruction

```
LD B, (IX+19H)
```

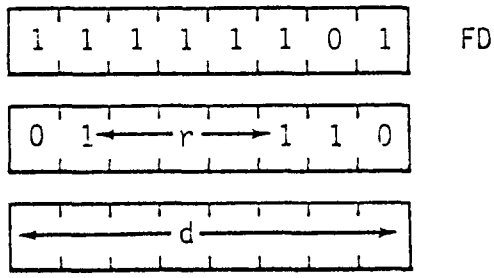
will cause the calculation of the sum 25AFH + 19H, which points to memory location 25C8H. If this address contains byte 39H, the instruction will result in register B also containing 39H.

LD r, (IY+d)

Operation: r ← (IY+d)

Format:

Opcode	Operands
LD	r, (IY+d)



Description:

The operand (IY+d) (the contents of the Index Register IY summed with a displacement integer d) is loaded into register r, where r identifies register A, B, C, D, E, H or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M CYCLES: 5 T STATES: 19(4,4,3,5,3)

Condition Bits Affected: None

Example:

If the Index Register IY contains the number 25AFH, the instruction

```
LD B, (IY+19H)
```

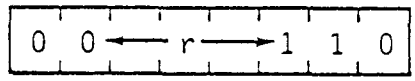
will cause the calculation of the sum 25AFH + 19H, which points to memory location 25C8H. If this address contains byte 39H, the instruction will result in register B also containing 39H.

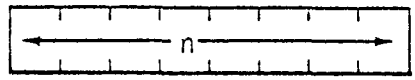
LD r, n

Operation: r ← n

Format:

Opcode	Operands
LD	r, n





Description:

The eight-bit integer n is loaded into any register r, where r identifies register A, B, C, D, E, H or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M CYCLES: 2 T STATES: 7(4,3)

Condition Bits Affected: None

Example:

After the execution of

```
LD E, A5H
```

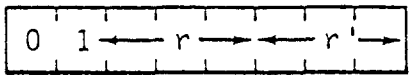
the contents of register E will be A5H.

LD r, r′

Operation: r ← r′

Format:

Opcode	Operands
LD	r,r′



Description:

The contents of any register r′ are loaded into any other register r. Note: r,r′ identifies any of the registers A, B, C, D, E, H, or L, assembled as follows in the object code:

Register	r,r′
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M CYCLES: 1 T STATES: 4

Condition Bits Affected: None

Example:

If the H register contains the number 8AH, and the E register contains 10H, the instruction

```
LD H, E
```

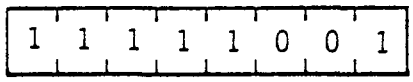
would result in both registers containing 10H.

LD SP, HL

Operation: SP ← HL

Format:

Opcode	Operands
LD	SP,HL



F9

Description.

The contents of the register pair HL are loaded into the Stack Pointer SP.

M CYCLES: 1 T STATES: 6

Condition Bits Affected: None

Example:

If the register pair HL contains 442EH, after the instruction

```
LD SP,HL
```

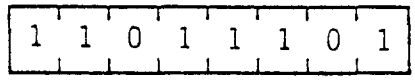
the Stack Pointer will also contain 442EH.

LD SP, IX

Operation: SP ← IX

Format:

Opcode	Operands
LD	SP,IX



DD

F9

Description:

The two byte contents of Index Register IX are loaded into the Stack Pointer SP.

M CYCLES: 2 T STATES: 10(4,6)

Condition Bits Affected: None

Example:

If the contents of the Index Register IX are 98DAH, after the instruction

LD SP,IX

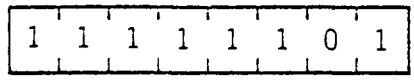
the contents of the Stack Pointer will also be 98DAH.

LD SP, IY

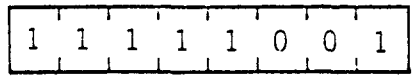
Operation: SP ← IY

Format:

Opcode	Operands
LD	SP, IY



FD



F9

Description:

The two byte contents of Index Register IY are loaded into the Stack Pointer SP.

M CYCLES: 2 T STATES: 10(4,6)

Condition Bits Affected: None

Example:

If Index Register IY contains the integer A227H, after the instruction

```
LD SP,IY
```

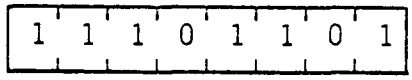
the Stack Pointer will also contain A227H.

LDD

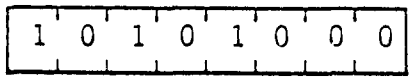
Operation: (DE) ← (HL), DE ← DE-1, HL ← HL-1, BC ← BC-1

Format:

Opcode	Operands
LDD	



ED



A8

Descripttion:

This two byte instruction transfers a byte of data from the memory location addressed by the contents of the HL register pair to the memory location addressed by the contents of the DE register pair. Then both of these register pairs including the BC (Byte Counter) register pair are decremented.

M CYCLES: 4 T STATES: 16(4,4,3,5)

Condition Bits Affected:

S:	Not affected
Z:	Not affected
H:	Reset
P/V:	Set if BC-1≠0; reset otherwise
N:	Reset
C:	Not affected

Example:

If the HL register pair contains 1111H, memory location 1111H contains the byte 88H, the DE register pair contains 2222H, memory location 2222H contains byte 66H, and the BC register pair contains 7H, then the instruction

```
LDD
```

will result in the following contents in register pairs and memory addresses:

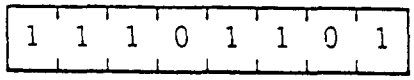
```
    HL  :  1110H
(1111H) :    88H
    DE  :  2221H
(2222H) :    88H
    BC  :     6H
```

LDDR

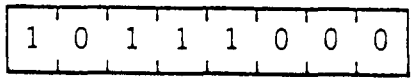
Operation: (DE) ← (HL), DE ← DE-1, HL ← HL-1, BC ← BC-1

Format:

Opcode	Operands
LDDR	



ED



B8

Description:

This two byte instruction transfers a byte of data from the memory location addressed by the contents of the HL register pair to the memory location addressed by the contents of the DE register pair. Then both of these registers as well as the BC (Byte Counter) are decremented. If decrementing causes the BC to go to zero, the instruction is terminated. If BC is not zero, the program counter is decremented by 2 and the instruction is repeated. Note that if BC is set to zero prior to instruction execution, the instruction will loop through 64K bytes. Interrupts will be recognized and two refresh cycles will be executed after each data transfer.

For BC≠0:

M CYCLES: 5 T STATES: 21(4,4,3,5,5)

For BC=0:

M CYCLES: 4 T STATES: 16(4,4,3,5)

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Reset
N: Reset
C: Not affected

Example:

If the HL register pair contains 1114H, the DE register pair contains 2225H, the BC register pair contains 0003H, and memory locations have these contents:

```
(1114H) : A5H     (2225H) : C5H
(1113H) : 36H     (2224H) : 59H
(1112H) : 88H     (2223H) : 66H
```

then after the execution of

```
LDDR
```

the contents of register pairs and memory locations will be:

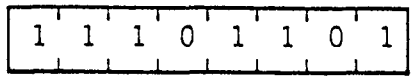
```
HL : 1111H
DE : 2222H
BC : 0000H

(1114H) : A5H     (2225H) : A5H
(1113H) : 36H     (2224H) : 36H
(1112H) : 88H     (2223H) : 88H
```

LDI

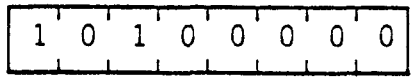
Operation: (DE) ← (HL), DE ← DE+1, HL ← HL+1, BC ← BC-1

Format:

Opcode	Operands
LDI	



ED



A0

Description:

A byte of data is transferred from the memory location addressed by the contents of the HL register pair to the memory location addressed by the contents of the DE register pair. Then both these register pairs are incremented and the BC (Byte Counter) register pair is decremented.

M CYCLES: 4 T STATES: 16(4,4,3,5)

Condition Bits Affected:

S:	Not affected
Z:	Not affected
H:	Reset
P/V:	Set if BC-1≠0; reset otherwise
N:	Reset
C:	Not affected

Example:

If the HL register pair contains 1111H, memory location 1111H contains contains the byte 88H, the DE register pair contains 2222H, the memory location 2222H contains byte 66H, and the BC register pair contains 7H, then the instruction

LDI

will result in the following contents in register pairs and memory addresses:

```
    HL    :   1112H
(1111H)   :   88H
    DE    : 2223H
(2222H)   :   88H
    BC    :    6H
```

LDIR

Operation: (DE) ← (HL), DE ← DE+1, HL ← HL+1, BC ← BC-1

Format:

Opcode	Operands	
LDIR		
1 1 1 0 1 1 0 1		ED
1 0 1 1 0 0 0 0		B0

Description:

This two byte instruction transfers a byte of data from the memory location addressed by the contents of the HL register pair to the memory location addressed by the DE register pair. Then both these register pairs are incremented and the BC (Byte Counter) register pair is decremented. If decrementing causes the BC to go to zero, the instruction is terminated. If BC is not zero the program counter is decremented by 2 and the instruction is repeated. Note that if BC is set to zero prior to instruction execution, the instruction will loop through 64K bytes. Interrupts will be recognized and two refresh cycles will be executed after each data transfer.

For BC≠0:

M CYCLES: 5 T STATES: 21(4,4,3,5,5)

For BC=0:

M CYCLES: 4 T STATES: 16(4,4,3,5)

Condition Bits Affected:

S:	Not affected
Z:	Not affected
H:	Reset
P/V:	Reset
N:	Reset
C:	Not affected

Example:

If the HL register pair contains 1111H, the DE register pair contains 2222H, the BC register pair contains 0003H, and memory locations have these contents:

(1111H)	: 88H	(2222H)	: 66H
(1112H)	: 36H	(2223H)	: 59H
(1113H)	: A5H	(2224H)	: C5H

then after the execution of

```
LDIR
```

the contents of register pairs and memory locations will be:

```
HL : 1114H
DE : 2225H
BC : 0000H
```

(1111H)	: 88H	(2222H)	: 88H
(1112H)	: 36H	(2223H)	: 36H
(1113H)	: A5H	(2224H)	: A5H

NEG

Operation: A ← o-A

Format:

Opcode								
NEG								
1	1	1	0	1	1	0	1	ED
0	1	0	0	0	1	0	0	44

Description:

Contents of the Accumulator are negated (two's complement). This is the same as subtracting the contents of the Accumulator from zero. Note that 80H is left unchanged.

M CYCLES: 2 T STATES: 8(4,4)

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if borrow from Bit 4; reset otherwise
P/V: Set if Acc, was 80H before operation; reset otherwise
N: Set
C: Set if Acc, was not OOH before ooperation; reset otherwise

Example:

If the contents of the Accumulator are

1	0	0	1	1	0	0	0

after the execution of

NEG

the Accumulator contents will be

0	1	1	0	1	0	0	0

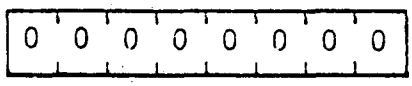
NOP

Operation: —

Format:

Opcode

NOP



00

Description:

CPU performs no operation during this machine cycle.

M CYCLES: 1 T STATES: 4

Condition Bits Affected: None

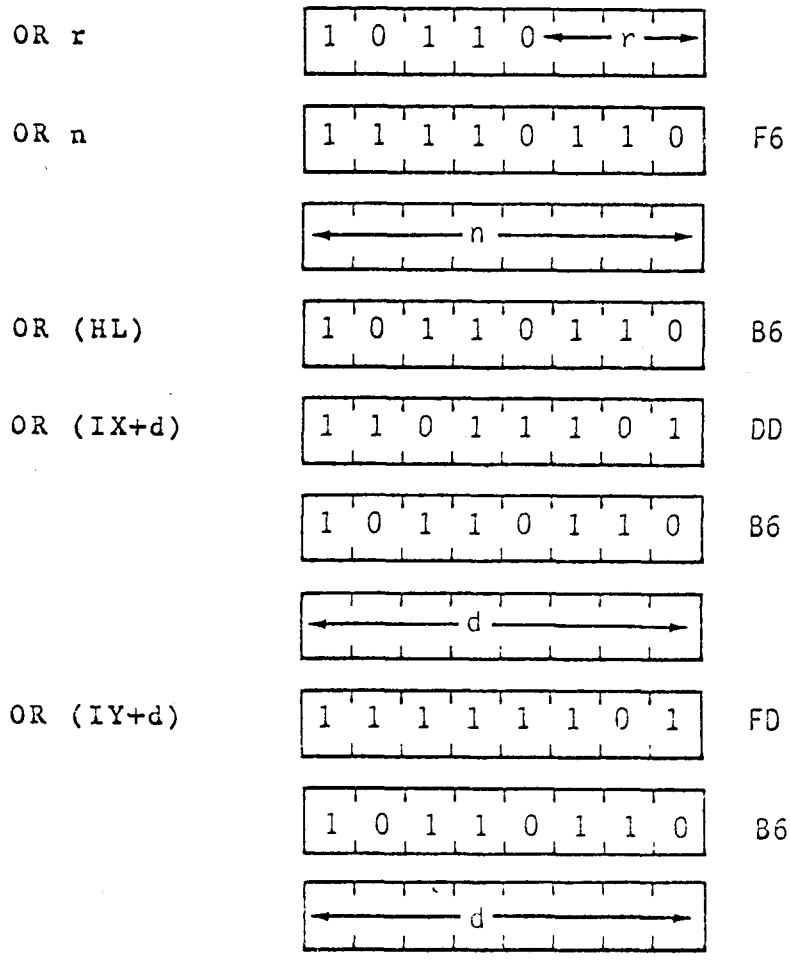
OR s

Operation: $A \leftarrow A \vee s$

Format:

Opcode	Operands
OR	s

The s operand is any of r,n,(HL),(IX+d) or (IY+d), as defined for the analogous ADD instructions. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B,C,D,E,H,L or A assembled as follows in the object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

A logical OR operation, bit by bit, is performed between the byte specified by the s operand and the byte contained in the Accumulator; the result is stored in the Accumulator.

INSTRUCTION	M CYCLES	T STATES
OR r	1	4
OR n	2	7(4,3)
OR (HL)	2	7(4,3)
OR (IX+d)	5	19(4,4,3,5,3)
OR (IY+d)	5	19(4,4,3,5,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set
P/V:	Set if parity even; reset otherwise
N:	Reset
C:	Reset

Example:

If the H register contains 48H (010001000) and the Accumulator contains 12H (00010010) after the execution of

OR H

the Accumulator will contain 5AH (01011010).

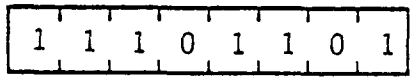
OTDR

Operation: (C) ← (HL), B ← B-1, HL ← HL-1

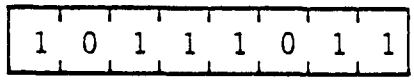
Format:

Opcode

OTDR



ED



BB

Description:

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its decremented value is placed on the top half (A8 through A15) of the address bus at this time. Next the byte to be output is placed on the data bus and written into the selected peripheral device. Then register pair HL is decremented and if the decremented B register is not zero, the Program Counter (PC) is decremented by 2 and the instruction is repeated. If B has gone to zero, the instruction is terminated. Note that if B is set to zero prior to instruction execution, the instruction will output 256 byte of data. Also, interrupts will be recognized after each data transfer.

If B≠0:

M CYCLES: 5 T STATES: 21(4,5,3,4,5)

If B=0:

M CYCLES: 4 T STATES: 16(4,5,3,4)

Condition Bits Affected:

S:	Unknown
Z:	Set
H:	Unknown
P/V:	Unknown
N:	Set
C:	unknown

Example:

If the contents of register C are 07H, the contents of register B are 03H, the contents of the HL register pair are 1000H, and memory locations have the following contents:

Location	Contents
0FFEH	51H
0FFFH	A9H
1000H	03H

then after the execution of

```
OTDR
```

the HL register pair will contain 0FFDH, register B will contain zero, and a group of bytes will have been written to the peripheral device mapped to I/O port addres 07H in the following sequence:

```
03H
A9H
51H
```

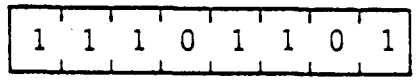
OTIR

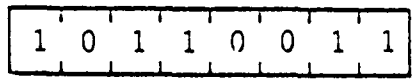
Operation: (C) ← (HL), B ← B-1, HL ← HL + 1

Format:

Opcode

OTIR





Description:

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its decremented value is placed on the top half (A8 through A15) of the address bus at this time. Next the byte to be output is placed on the data bus and written into the selected peripheral device. Then register pair HL is incremented. If the decremented B register is not zero, the Program Counter (PC) is decremented by 2 and the instruction is repeated. If B has gone to zero, the instruction is terminated. Note that if B is set to zero prior to instruction execution, the instruction will output 256 bytes of data. Interrupts will be recognized and two refresh cycles will be executed after each data transfer.

If B≠0:

M CYCLES: 5 T STATES: 21(4,5,3,4,5)

If B=0:

M CYCLES: 4 T STATES: 16(4,5,3,4)

Condition Bits Affected:

S:	Unknown
Z:	Set
H:	Unknown
P/V:	Unknown
N:	Set
C:	unknown

Example:

If the contents of register C are 07H, the contents of register B are 03H, the contents of the HL register pair are 1000H, and memory locations have the following contents:

Location	Contents
1000H	51H
1001H	A9H
1002H	03H

then after the execution of

```
OTIR
```

the HL register pair will contain 1003H, register B will contain zero, and a group of bytes will have been written to the peripheral device mapped to I/O port address 07H in the following sequence:

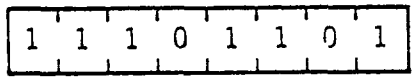
```
51H
A9H
03H
```

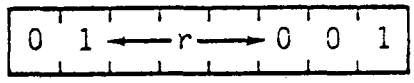
OUT (C), r

Operation: (C) ← r

Format:

Opcode	Operands
OUT	(C),r





Description:

The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. The contents of Register B are placed on the top half (A8 through A15) of the address bus at this time. Then the byte contained in register r is placed on the data bus and written into the selected peripheral device. Register r identifies any of the CPU registers shown in the following table, which also shows the corresponding 3-bit "r" field for each which appears in the assembled object code:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

M CYCLES: 3 T STATES: 12(4,4,4)

Condition Bits Affected: None

Example:

If the contents of register C are 01H and the contents of register D are 5AH, after the execution of

```
OUT (C),D
```

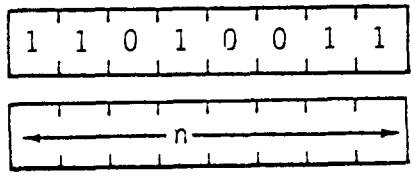
the byte 5AH will have been written to the peripheral device mapped to I/O port address 01H.

OUT (n), A

Operation: (n) ← A

Format:

Opcode	Operands
OUT	(n),A



Description:

The operand n is placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. The contents of the Accumulator (register A) also appear on the top half (A8 through A15) of the address bus at this time. Then the byte contained in the Accumulator is placed on the data bus and written into the selected peripheral device.

M CYCLES: 3 T STATES: 11(4,3,4)

Condition Bits Affected: None

Example:

If the contents of the Accumulator are 23H, then after the execution of

```
OUT 01H,A
```

the byte 23H will have been written to the peripheral device mapped to I/O port address 01H.

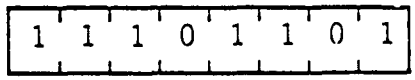
OUTD

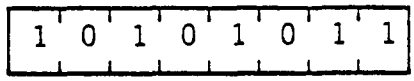
Operation: (C) ← (HL), B ← B-1, HL ← HL-1

Format:

Opcode

OUTD





Description:

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its decremented value is placed on the top half (A8 through A15) of the address bus at this time. Next the byte to be output is placed on the data bus and written into the selected peripheral device. Finally the register pair HL is incremented.

M CYCLES: 4 T STATES: 16(4,5,3,4)

Condition Bits Affected:

S:	Unknown
Z:	Set if B-1=0; reset otherwise
H:	Unknown
P/V:	Unknown
N:	Set
C:	unknown

Example:

If the contents of register C are 07H, the contents of

register B are 10H, the contents of the HL register pair are 1000H, and the contents of memory location 1000H are 59H, after the execution of

```
OUTD
```

register B will contain 0FH, the HL register pair will contain 0FFFH, and the byte 59H will have been written to the peripheral device mapped to I/O port address 07H.

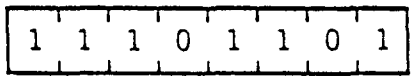
OUTI

Operation: (C) ← (HL), B ← B-1, HL ← HL + 1

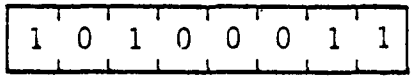
Format:

Opcode

OUTI



ED



A3

Description:

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its decremented value is placed on the top half (A8 through A15) of the address bus. The byte to be output is placed on the data bus and written into selected peripheral device. Finally the register pair HL is incremented.

M CYCLES: 4 T STATES: 16(4,5,3,4)

Condition Bits Affected:

S:	Unknown
Z:	Set if B-1=0; reset otherwise
H:	Unknown
P/V:	Unknown
N:	Set
C:	unknown

Example:

If the contents of register C are 07H, the contents of register B are 10H, the contents of the HL register pair are 1000H, and the contents of memory address 1000H are

59H, then after the execution of

OUTI

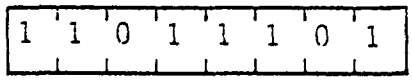
register B will contain 0FH, the HL register pair will contain 1001H, and the byte 59H will have been written to the peripheral device mapped to I/O port address 07H.

POP IX

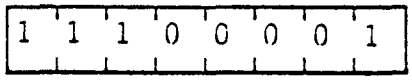
Operation: $IX_H \leftarrow (SP+1)$, $IX_L \leftarrow (SP)$

Format:

Opcode	Operands
POP	IX



DD



E1

Description:

The top two bytes of the external memory LIFO (last-in, first-out) Stack are popped into Index Register IX. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first loads into the low order portion of IX the byte at the memory location corresponding to the contents of SP; then SP is incremented and the contents of the corresponding adjacent memory location are loaded into the high order portion of IX. The SP is now incremented again.

M CYCLES: 4 T STATES: 14(4,4,3,3)

Condition Bits Affected: None

Example:

If the Stack Pointer contains 1000H, memory location 1000H contains 55H, and location 1001H contains 33H; the instruction

```
POP IX
```

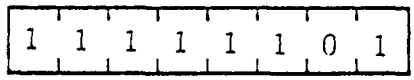
will result in Index Register IX containing 3355H, and the Stack Pointer containing 1002H.

POP IY

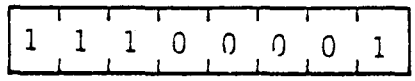
Operation: $IY_H \leftarrow (SP+1)$, $IY_L \leftarrow (SP)$

Format:

Opcode	Operands
POP	IY



FD



E1

Description:

The top two bytes of the external memory LIFO (last-in, first-out) Stack are popped into Index Register IY. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first loads into the low order portion of IY the byte at the memory location corresponding to the contents of SP; then SP is incremented and the contents of the corresponding adjacent memory location are loaded into the high order portion of IY. The SP is now incremented again.

M CYCLES: 4 T STATES: 14(4,4,3,3)

Condition Bits Affected: None

Example:

If the Stack Pointer contains 1000H, memory location 1000H contains 55H, and location 1001H contains 33H, the instruction

POP IY

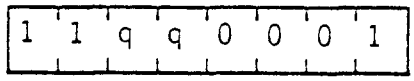
will result in Index Register IY containing 3355H, and the Stack Pointer containing 1002H.

POP qq

Operation: $qq_H \leftarrow (SP+1)$, $qq_L \leftarrow (SP)$

Format:

Opcode	Operands
POP	qq



Description:

The top two bytes of the external memory LIFO (last-in, first-out) Stack are popped into register pair qq. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first loads into the low order portion of qq, the byte at the memory location corresponding to the contents of SP; then SP is incremented and the contents of the corresponding adjacent memory location are loaded into the high order portion of qq and the SP is now incremented again. The operand qq defines register pair BC, DE, HL, or AF, assembled as follows in the object code:

Pair	r
BC	00
DE	01
HL	10
AF	11

M CYCLES: 3 T STATES: 10(4,3,3)

Condition Bits Affected: None

Example:

If the Stack Pointer contains 1000H, memory location 1000H contains 55H, and location 1001H contains 33H, the instruction

```
POP HL
```

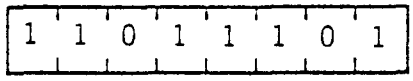
will result in register pair HL containing 3355H, and the Stack Pointer containing 1002H.

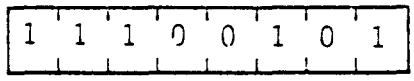
PUSH IX

Operation: $(SP-2) \leftarrow IX_L$, $(SP-1) \leftarrow IX_H$

Format:

Opcode	Operands
PUSH	IX





Description:

The contents of the Index Register IX are pushed into the external memory LIFO (last-in, first-out) Stack. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first decrements the SP and loads the high order byte of IX into the memory address now specified by the SP; then decrements the SP again and loads the low order byte into the memory location corresponding to this new address in the SP.

M CYCLES: 3 T STATES: 15(4,5,3,3)

Condition Bits Affected: None

Example:

If the Index Register IX contains 2233H and the Stack Pointer contains 1007H, after the instruction

PUSH IX

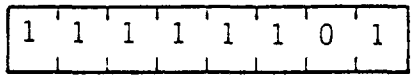
memory address 1006H will contain 22H, memory address 1005H will contain 33H, and the Stack Pointer will contain 1005H.

PUSH IY

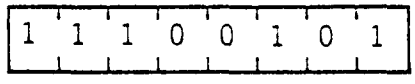
Operation: $(SP-2) \leftarrow IY_L$, $(SP-1) \leftarrow IY_H$

Format:

Opcode	Operands
PUSH	IY



FD



E5

Description:

The contents of the Index Register IY are pushed into the external memory LIFO (last-in, first-out) Stack. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first decrements the SP and loads the high order byte of IY into the memory address now specified by the SP; then decrements the SP again and loads the low order byte into the memory location corresponding to this new address in the SP.

M CYCLES: 4 T STATES: 15(4,5,3,3)

Condition Bits Affected: None

Example:

If the Index Register IY contains 2233H and the Stack Pointer contains 1007H, after the instruction

PUSH IY

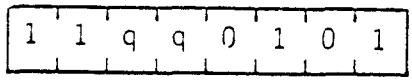
memory address 1006H will contain 22H, memory address 1005H will contain 33H, and the Stack Pointer will contain 1005H.

PUSH qq

Operation: $(SP-2) \leftarrow qq_L$, $(SP-1) \leftarrow qq_H$

Format:

Opcode	Operands
PUSH	qq



Description:

The contents of the register pair qq are pushed into the external memory LIFO (last-in, first-out) Stack. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first decrements the SP and loads the high order byte of register pair qq into the memory address now specified by the SP; then decrements the SP again and loads the low order byte of qq into the memory location corresponding to this new address in the SP. The operand qq means register pair BC, DE, HL, or AF, assembled as follows in the object code:

Pair	qq
BC	00
DE	01
HL	10
AF	11

M CYCLES: 3 T STATES: 11(5,3,3)

Condition Bits Affected: None

Example:

If the AF register pair contains 2233H and the Stack Pointer contains 1007H, after the instruction

PUSH AF

memory address 1006H will contain 22H, memory address 1005H will contain 33H, and the Stack Pointer will contain 1005H.

RES b, m

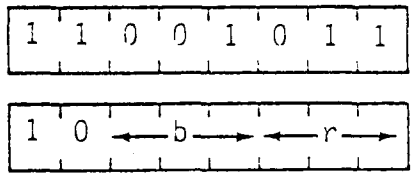
Operation: $s_b \leftarrow 0$

Format:

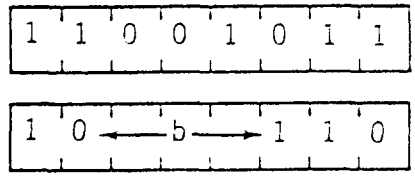
Opcode	Operands
RES	b,m

Operand b is any bit (7 through 0) of the contents of the m operand, (any of r, (HL), (IX+d) or (IY+d)) as defined for the analogous SET instructions. These various possible opcode-operand combinations are assembled as follows in the object code:

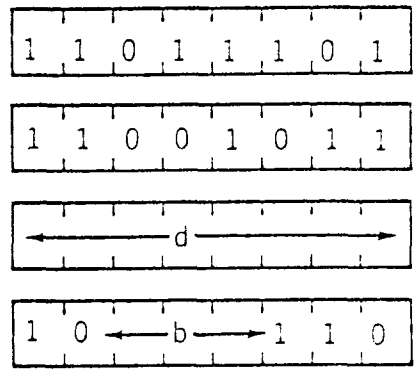
RES b,r



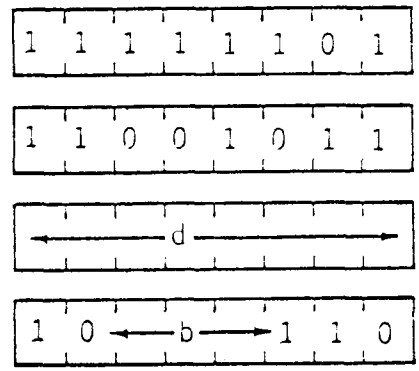
RES b,(HL)



RES b,(IX+d)



RES b,(IY+d)



Bit Reset	b	Register	r
0	000	B	000
1	001	C	001
2	010	D	010
3	011	E	011
4	100	H	100
5	101	L	101
6	110	A	111
7	111		

Description:

Bit b in operand m is reset.

INSTRUCTION	M CYCLES	T STATES
RES r	4	8(4,4)
RES (HL)	4	15(4,4,4,3)
RES (IX+d)	6	23(4,4,3,5,4,3)
RES (IY+d)	6	23(4,4,3,5,4,3)

Condition Bits Affected: None

Example:

After the execution of

```
RES 6,D
```

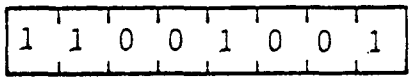
bit 6 in register D will be reset. (Bit 0 in register D is the least significant bit.)

RET

Operation: $PC_L \leftarrow (SP)$, $PC_H \leftarrow (SP+1)$

Format:

Opcode	
RET	



C9

Description:

Control is returned to the original program flow by popping the previous contents of the Program Counter (PC) off the top of the external memory stack, where they were pushed by the CALL instruction. This is accomplished by first loading the low-order byte of the PC with the contents of the memory address pointed to by the Stack Pointer (SP), then incrementing the SP and loading the high-order byte of the PC with the contents of the memory address now pointed to by the SP. (The SP is now incremented a second time.) On the following machine cycle the CPU will fetch the next program opcode from the location in memory now pointed to by the PC.

M CYCLES: 3 T STATES: 10(4,3,3)

Condition Bits Affected: None

Example:

If the contents of the Program Counter are 3535H, the contents of the Stack Pointer are 2000H, the contents of memory location 2000H are B5H, and the contents of memory location 2001H are 18H, then after the execution of

RET

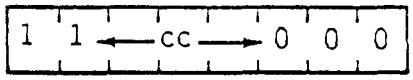
the contetns of the Stack Pointer will be 2002H and the contents of the Program Counter will be 18B5H, pointing to the address of the next program opcode to be fetched.

RET cc

Operation: IF cc TRUE: $PC_L \leftarrow (SP), PC_H \leftarrow (SP+1)$

Format:

Opcode	Operand
RET	cc



Description:

If condition cc is true, control is returned to the original program flow by popping the previous contents of the Program Counter (PC) off the top of the external memory stack, where they were pushed by the CALL instruction. This is accomplished by first loading the low-order byte of the PC with the contents of the memory address pointed to by the Stack Pointer (SP), then incrementing the SP, and loading the high-order byte of the PC with the contents of the memory address now pointed to by the SP. (The SP is now incremented a second time.) On the following machine cycle the CPU will fetch the next program opcode from the location in memory now pointed to by the PC. If condition cc is false, the PC is simply incremented as usual, and the program continues with the next sequential instruction. Condition cc is programmed as one of eight status which correspond to condition bits in the Flag Register (register F). These eight status are defined in the table below, which also specifies the corresponding cc bit fields in the assembled object code.

cc	Condition	Relevant Flag
000	NZ non zero	Z
001	Z zero	Z
010	NC non carry	C
011	C carry	C
100	PO parity odd	P/V
101	PE parity even	P/V
110	P sign positive	S
111	M sign negative	S

If cc is true:

M CYCLES: 3 T STATES: 11(5,3,3)

If cc is false:

M CYCLES: 1 T STATES: 5

Condition Bits Affected: None

Example:

If the S flag in the F register is set, the contents of the Program Counter are 3535H, the contents of the Stack Pointer are 2000H, the contents of memory location 2000H are B5H, and the contents of memory location 2001H are 18H, then after the execution of

```
RET M
```

the contents of the Stack Pointer will be 2002H and the contents of the Program Counter will be 18B5H, pointing to the address of the next program opcode to be fetched.

RETI

Operation: Return from interrupt

Format:

Opcode

RETI

1	1	1	0	1	1	0	1	ED
0	1	0	0	1	1	0	1	4D

Description:

This instruction is used at the end of an interrupt service routine to:

1. Restore the contents of the Program Counter (PC) (analogous to the RET instruction)

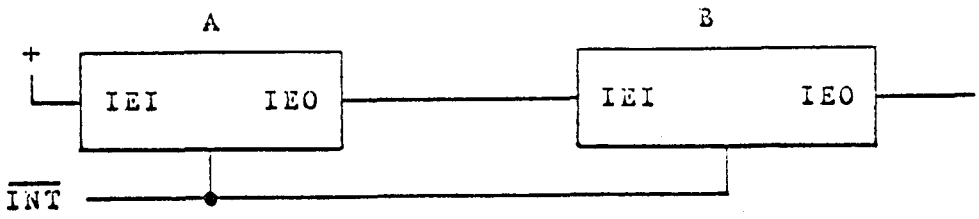
2. To signal an I/O device that the interrupt routine has been completed. The RETI instruction facilitates the nesting of interrupts allowing higher priority devices to suspend service of lower priority service routines. The state of IFF2 is copied into IFF1.

M CYCLES: 4 T STATES: 14(4,4,3,3)

Condition Bits Affected: None

Example:

Given: Two interrupting devices, A and B connected in a daisy chain configuration with A having a higher priority than B.



B generates an interrupt and is acknowledged. (The interrupt enable out, IEO, of B goes low, blocking any lower priority devices from interrupting while B is being serviced). Then A generates an interrupt, suspending service of B. (The IEO of A goes 'low' indicating that a higher priority device is being serviced.) The A routine is completed and a RETI is issued resetting the IEO of A, allowing the B routine to continue. A second RETI is issued on completion of the B routine and the IEO of B is reset (high) allowing lower priority devices interrupt access.

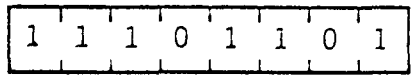
RETN

Operation: Return from non maskable interrupt

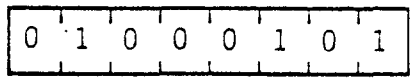
Format:

Opcode

RETN



ED



45

Description:

Used at the end of a service routine for a non maskable interrupt, this instruction executes an unconditional return which functions identical to the RET instruction. That is, the previously stored contents of the Program Counter (PC) are popped off the top of the external memory stack; the low-order byte of PC is loaded with the contents of the memory location pointed to by the Stack Pointer (SP), SP is incremented, the high-order byte of PC is loaded with the contents of the memory location now pointed to by SP, and SP is incremented again. Control is now returned to the original program flow: on the following machine cycle the CPU will fetch the next opcode from the location in memory now pointed to by the PC. Also the state of IFF2 is copied back into IFF1 to the state it had prior to the acceptance of the NMI.

M CYCLES: 4 T STATES: 14(4,4,3,3)

Condition Bits Affected: None

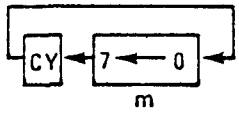
Example:

If the contents of the Stack Pointer are 1000H and the contents of the Program Counter are 1A45H when a non maskable interrupt (NMI) signal is received, the CPU will ignore the next instruction and will instead restart to memory address 0066H. That is, the current Program Counter contents of 1A45H will be pushed onto the external stack address of 0FFFH and 0FFEH, high

order-byte first, and 0066H will be loaded onto the Program Counter. That address begins an interrupt service routine which ends with RETN instruction. Upon the execution of RETN, the former Program Counter contents are popped off the external memory stack, low-order first, resulting in a Stack Pointer contents again of 1000H. The program flow continues where it left off with an opcode fetch to address 1A45H.

RL m

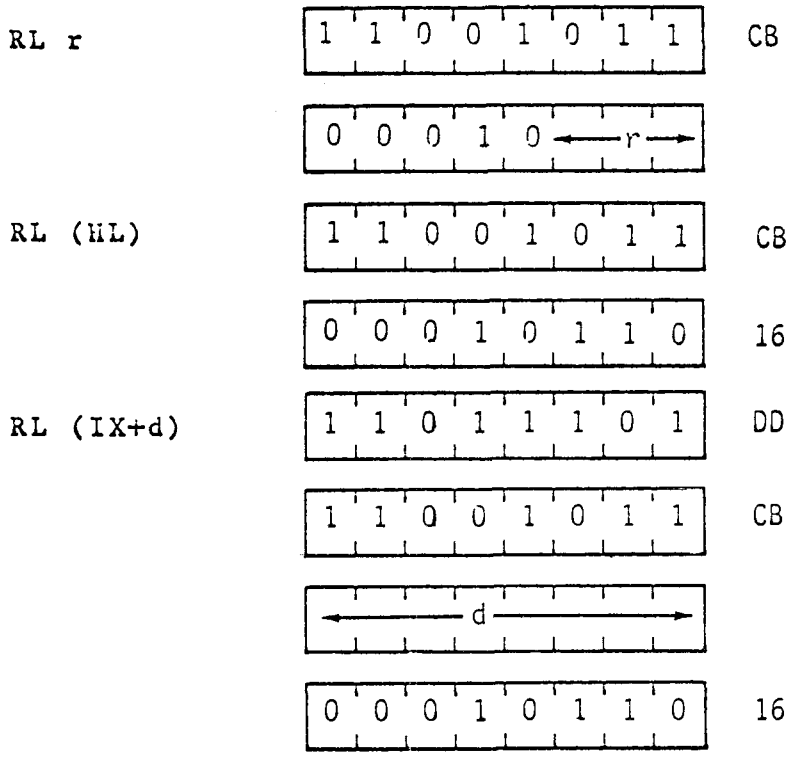
Operation:



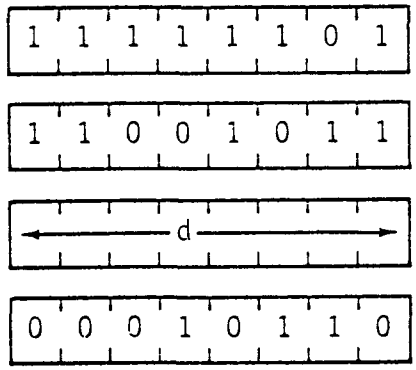
Format:

Opcode	Operands
RL	m

The m operand is any of r,(HL), (IX+d) or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:



RL (IY+d)



*r identifies registers B,C,D,E,H,L or A specified as follows in the assembled object code above:

Register	r
B	000
C	001
D	010
E	011
H	011
L	101
A	111

Description:

The contents of the m operand are rotated left: the content of bit 0 is copied into bit 1; the previous content of bit 1 is copied into bit 2; this pattern is continued throughout the byte. The content of bit 7 is copied into the Carry Flag (C flag in register F) and the previous content of the Carry Flag is copied into bit 0 (Bit 0 is the least significant bit.)

INSTRUCTION	M CYCLES	T STATES
RL r	2	8(4,4)
RL (HL)	4	15(4,4,4,3)
RL (IX+d)	6	23(4,4,3,5,4,3)
RL (IY+d)	6	23(4,4,3,5,4,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Reset
P/V:	Set if parity even; reset otherwise
N:	Reset
C:	Data from Bit 7 of source register

Example:

If the contents of register D and the Carry Flag are

C	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	1	1

after the execution of

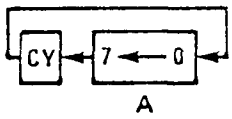
```
RL D
```

the contents of register D and the Carry Flag will be

C	7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	1	0

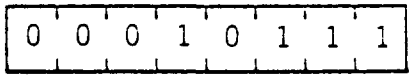
RLA

Operation:



Format:

Opcode	Operands
RLA	



17

Description:

The contents of the Accumulator (register A) are rotated left: the content of bit 0 is copied into bit 1; the previous content of bit 1 is copied into bit 2; this pattern is continued throughout the register. The content of bit 7 is copied into the Carry Flag (C flag in register F) and the previous content of the Carry Flag is copied into bit 0. Bit 0 is the least significant bit.

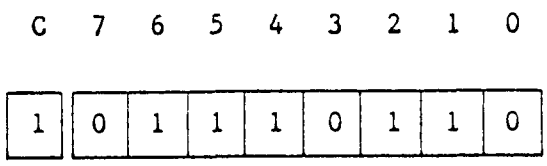
M CYCLES: 1 T STATES: 4

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Not affected
N: Reset
C: Data from Bit 7 of Acc.

Example:

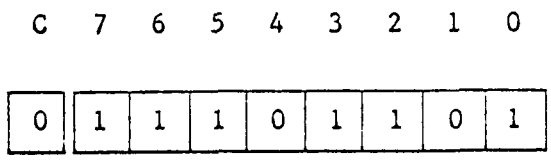
If the contents of the Accumulator and the Carry Flag are



after the execution of

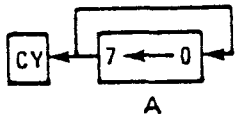
RLA

the contents of the Accumulator and the Carry Flag will be



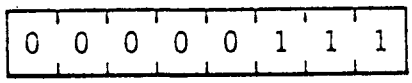
RLCA

Operation:



Format:

Opcode	Operands
RLCA	



07

Description:

The contents of the Accumulator (register A) are rotated left: the content of bit 0 is moved to bit 1; the previous content of bit 1 is moved to bit 2; this pattern is continued throughout the register. The content of bit 7 is copied into the Carry Flag (C flag in register F) and also into bit 0. (Bit 0 is the least significant bit.)

M CYCLES: 1 T STATES:4

Condition Bits Affected:

S:	Not affected
Z:	Not affected
H:	Reset
P/V:	Not affected
N:	Reset
C:	Data from Bit 7 of Acc.

Example:

If the contents of the Accumulator are

7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0

after the execution of

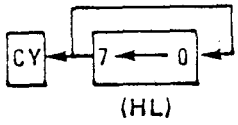
RLCA

the contents of the Accumulator and Carry Flag will be

C	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1

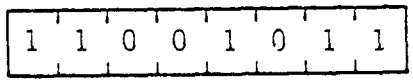
RLC (HL)

Operation:

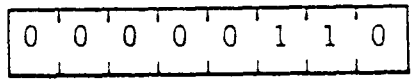


Format:

Opcode	Operands
RLC	(HL)



CB



06

Description:

The contents of the memory address specified by the contents of register pair HL are rotated left: the content of bit 0 is copied into bit 1; the previous content of bit 1 is copied into bit 2; this pattern is continued throughout the byte. The content of bit 7 is copied into the Carry Flag (C flag in register F) and also into bit 0. Bit 0 is the least significant bit.

M CYCLES: 4 T STATES: 15(4,4,4,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Reset
P/V:	Set if parity even; reset otherwise
N:	Reset
C:	Data from Bit 7 of source register

Example:

If the contents of the HL register pair are 2828H, and the contents of memory location 2828H are

7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0

after the execution of

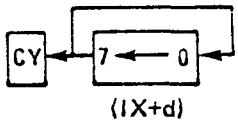
RLC (HL)

the contents of memory location 2828H and the Carry Flag will be

C	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1

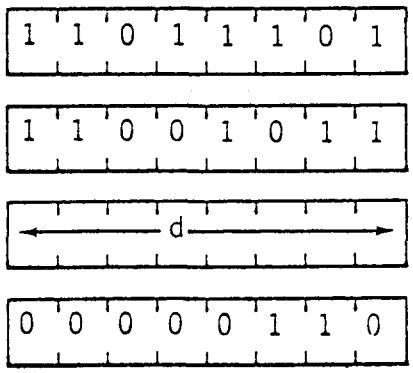
RLC (IX+d)

Operation:



Format:

Opcode	Operands
RLC	(IX+d)



Description:

The contents of the memory address specified by the sum of the contents of the Index Register IX and a two's complement displacement integer d, are rotated left: the contents of bit 0 is copied into bit 1; the previous content of bit 1 is copied into bit 2; this pattern is continued throughout the byte. The content of bit 7 is copied into the Carry Flag (C flag in register F) and also into bit 0. Bit 0 is the least significant bit.

M CYCLES: 6 T STATES: 23(4,4,3,5,4,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Reset
P/V:	Set if parity even; reset otherwise
N:	Reset
C:	Data from Bit 7 of source register

Example:

If the contents of the Index Register IX are 1000H, and the contents of memory location 1022H are

7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0

after the execution of

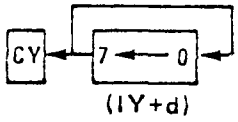
RLC (IX+2H)

the contents of memory location 1002H and the Carry Flag will be

C	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1

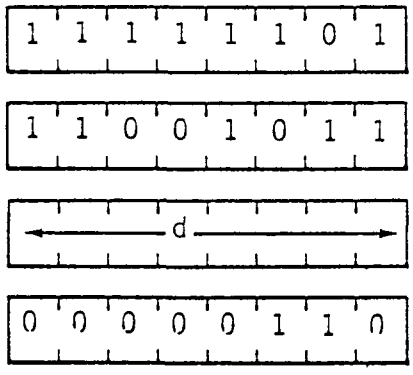
RLC (IY+d)

Operation:



Format:

Opcode	Operands
RLC	(IY+d)



Description:

The contents of the memory address specified by the sum of the contents of the Index Register IY and a two's complement displacement integer d are rotated left: the content of bit 0 is copied into bit 1; the previous content of bit 1 is copied into bit 2; this process is continued throughout the byte. The content of bit 7 is copied into the Carry Flag (C flag in register F) and also into bit 0. Bit 0 is the least significant bit.

M CYCLES: 6 T STATES: 23(4,4,3,5,4,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Reset
P/V:	Set if parity even; reset otherwise
N:	Reset
C:	Data from Bit 7 of source register

Example:

If the contents of the Index Register IY are 1000H, and the contents of memory location 1002H are

7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0

after the execution of

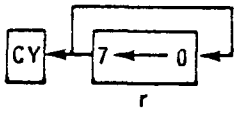
RLC (IY+2H)

the contents of memory location 1002H and the Carry Flag will be

C	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1

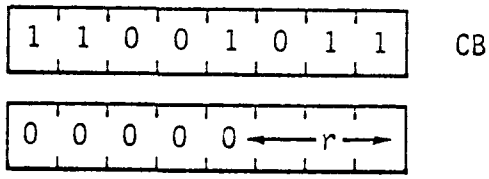
RLC r

Operation:



Format:

Opcode	Operands
RLC	r



Description:

The eight-bit contents of register r are rotated left: the content of bit 0 is copied into bit 1; the previous content of bit 1 is copied into bit 2; this pattern is continued throughout the register. The content of bit 7 is copied into the Carry Flag (C flag in register F) and also into bit 0. Operand r is specified as follows in the assembled object code:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Note: Bit 0 is the least significant bit.

M CYCLES: 2 T STATES: 8(4,4)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Reset
P/V:	Set if parity even; reset otherwise
N:	Reset
C:	Data from Bit 7 of source register

Example:

If the contents of register r are

7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0

after the execution of

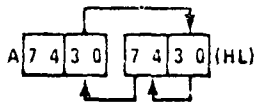
RLC r

the contents of register r and the Carry Flag will be

C	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1

RLD

Operation:



Format:

Opcode	Operands	
RLD		
1 1 1 0 1 1 0 1		ED
0 1 1 0 1 1 1 1		6F

Description:

The contents of the low order four bits (bits 3,2,1 and 0) of the memory location (HL) are copied into the high order four bits (7,6,5 and 4) of that same memory location; the previous contents of those high order four bits are copied into the low order four bits of the Accumulator (register A); and the previous contents of the low order four bits of the Accumulator are copied into the low order four bits of memory location (HL). The contents of the high order bits of the Accumulator are unaffected. Note: (HL) means the memory location specified by the contents of the HL register pair.

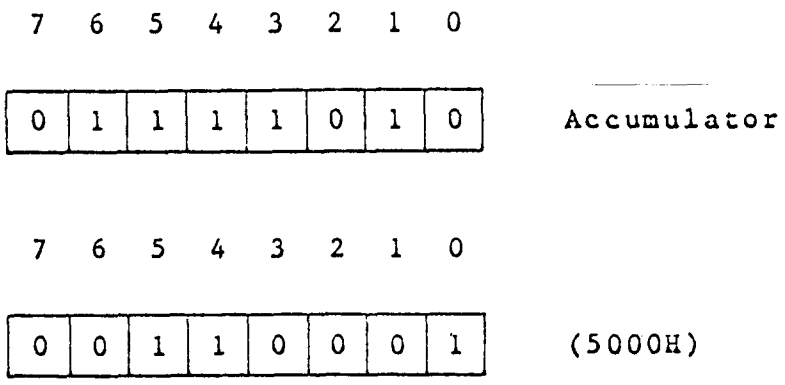
M CYCLES: 5 T STATES: 18(4,4,3,4,3)

Condition Bits Affected:

S:	Set if Acc. is negative after operation; reset otherwise
Z:	Set if Acc. is zero after operation; reset otherwise
H:	Reset
P/V:	Set if parity of Acc. is even after operation; reset otherwise
N:	Reset
C:	Not affected

Example:

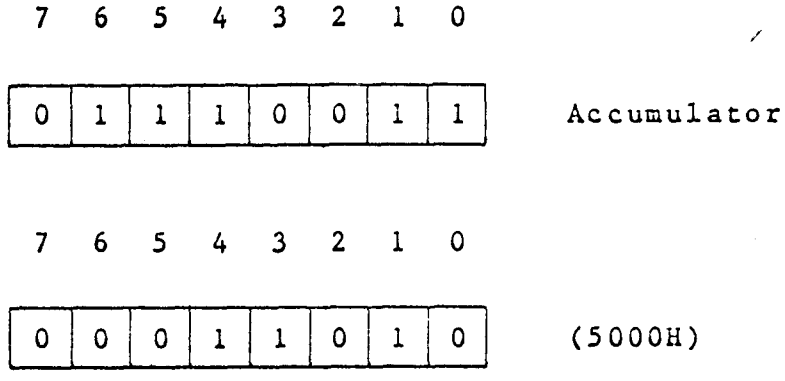
If the contents of the HL register pair are 5000H, and the contents of the Accumulator and memory location 5000H are



after the execution of

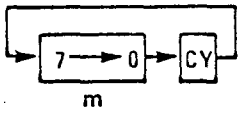
RLD

the contents of the Accumulator and memory location 5000H will be



RR m

Operation:

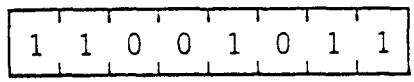


Format:

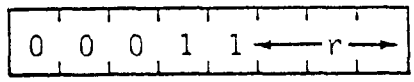
Opcode	Operand
RR	m

The m operand is any of r, (HL), (IX+d), or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:

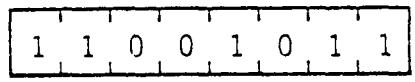
RR r



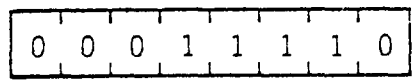
CB



RR (HL)

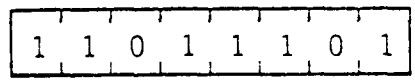


CB

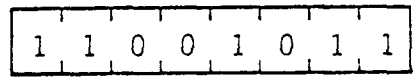


1E

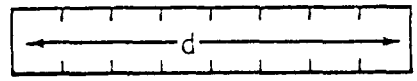
RR (IX+d)

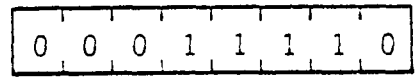


DD



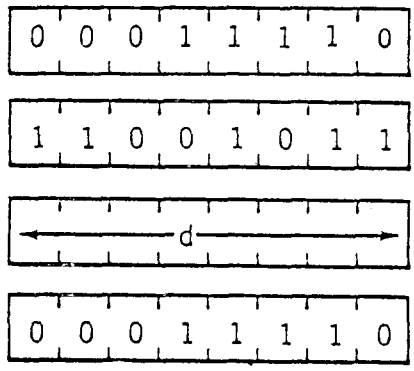
CB





1E

RR (IY+d)



*r identifies registers B,C,D,E,H,L or A specified as follows in the assembled object code above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

The contents of operand m are rotated right: the contents of bit 7 is copied into bit 6; the previous content of bit 6 is copied into bit 5; this pattern is continued throughout the byte. The content of bit 0 is copied into the Carry Flag (C flag in register F) and the previous content of the Carry Flag is copied into bit 7. Bit 0 is the least significant bit.

INSTRUCTION	M CYCLES	T STATES
RR r	2	8(4,4)
RR (HL)	4	15(4,4,4,3)
RR (IX+d)	6	23(4,4,3,5,4,3)
RR (IY+d)	6	23(4,4,3,5,4,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Reset
P/V:	Set if parity is even; reset otherwise
N:	Reset
C:	Data from Bit 0 of source register

Example:

If the contents of the HL register pair are 4343H, and the contents of memory location 4343H and the Carry Flag are

7	6	5	4	3	2	1	0	C
1	1	0	1	1	1	0	1	0

after the execution of

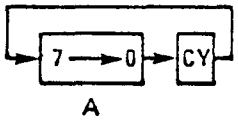
RR (HL)

the contents of location 4343H and the Carry Flag will be

7	6	5	4	3	2	1	0	C
0	1	1	0	1	1	1	0	1

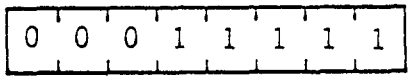
RRA

Operation:



Format:

Opcode	Operands
RRA	



1F

Description:

The contents of the Accumulator (register A) are rotated right: the content of bit 7 is copied into bit 6; the previous content of bit 6 is copied into bit 5; this pattern is continued throughout the register. The content of bit 0 is copied into the Carry Flag (C flag in register F) and the previous content of the Carry Flag is copied into bit 7. Bit 0 is the least significant bit.

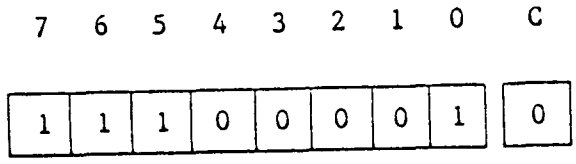
M CYCLES: 1 T STATES: 4

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Not affected
N: Reset
C: Data from Bit 0 of Acc.

Example:

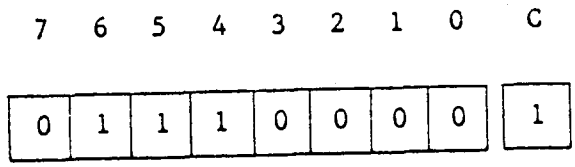
If the contents of the Accumulator and the Carry Flag are



after the execution of

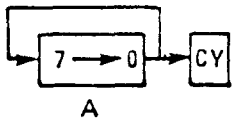
RRA

the contents of the Accumulator and the Carry Flag will be



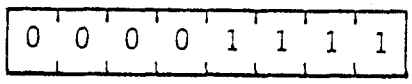
RRCA

Operation:



Format:

Opcode	Operands
RRCA	



0F

Description:

The contents of the Accumulator (register A) is rotated right: the content of bit 7 is copied into bit 6; the previous content of bit 6 is copied into bit 5; this pattern is continued throughout the register. The content of bit 0 is copied into bit 7 and also into the Carry Flag (C flag in register F.) Bit 0 is the least significant bit.

M CYCLES: 1 T STATES: 4

Condition Bits Affected:

S:	Not affected
Z:	Not affected
H:	Reset
P/V:	Not affected
N:	Reset
C:	Data from Bit 0 of Acc.

Example:

If the contents of the Accumulator are

7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	1

After the execution of

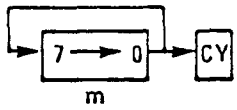
RRCA

the contents of the Accumulator and the Carry Flag will be

7	6	5	4	3	2	1	0	C
1	0	0	0	1	0	0	0	1

RRC m

Operation:



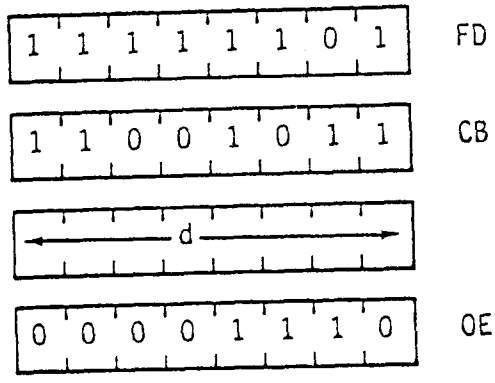
Format:

Opcode	Operands
RRC	m

The m operand is any of r,(HL), (IX+d) or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:

RRC r	1 1 0 0 1 0 1 1	CB
	0 0 0 0 1 ← r →	
RRC (HL)	1 1 0 0 1 0 1 1	CB
	0 0 0 0 1 1 1 0	0E
RRC (IX+d)	1 1 0 1 1 1 0 1	DD
	1 1 0 0 1 0 1 1	CB
	← d →	
	0 0 0 0 1 1 1 0	0E

RRC (IY+d)



*r identifies registers B,C,D,E,H,L or A specified as follows in the assembled object code above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

The contents of operand m are rotated right: the content of bit 7 is copied into bit 6; the previous content of bit 6 is copied into bit 5; this pattern is continued throughout the byte. The content of bit 0 is copied into the Carry Flag (C flag in the F register) and also into bit 7. Bit 0 is the least significant bit.

INSTRUCTION	M CYCLES	T STATES
RRC r	2	8(4,4)
RRC (HL)	4	15(4,4,4,3)
RRC (IX+d)	6	23(4,4,3,5,4,3)
RRC (IY+d)	6	23(4,4,3,5,4,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Reset
P/V:	Set if parity even; reset otherwise
N:	Reset
C:	Data from Bit 0 of source register

Example:

If the contents of register A are

7	6	5	4	3	2	1	0
0	0	1	1	0	0	0	1

after the execution of

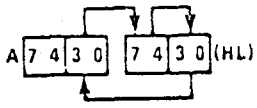
```
RRC A
```

the contents of register A and the Carry Flag will be

7	6	5	4	3	2	1	0	C
1	0	0	1	1	0	0	0	1

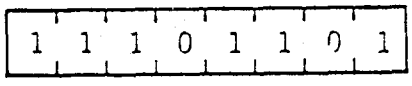
RRD

Operation:

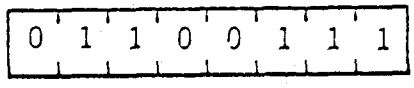


Format:

Opcode	Operands
RRD	



ED



67

Description:

The contents of the low order four bits (bits 3,2,1 and 0) of memory location (HL) are copied into the low order four bits of the Accumulator (register A); the previous contents of the low order four bits of the Accumulator are copied into the high order four bits (7,6,5 and 4) of location (HL); and the previous contents of the high order four bits of (HL) are copied into the low order four bits of (HL). The contents of the high order bits of the Accumulator are unaffected. Note: (HL) means the memory location specified by the contents of the HL register pair.

M CYCLES: 5 T STATES: 18(4,4,3,4,3)

Condition Bits Affected:

S:	Set if Acc. is negative after operation; reset otherwise
Z:	Set if Acc. is zero after operation; reset otherwise
H:	Reset
P/V:	Set if parity of Acc. is even after operation; reset otherwise
N:	Reset
C:	Not affected

Example:

If the contents of the HL register pair are 5000H, and the contents of the Accumulator and memory location 5000H are

7	6	5	4	3	2	1	0	
1	0	0	0	0	1	0	0	Accumulator

7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	0	(5000H)

after the execution of

RRD

the contents of the Accumulator and memory location 5000H will be

7	6	5	4	3	2	1	0	
1	0	0	0	0	0	0	0	Accumulator

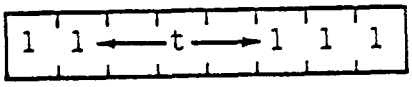
7	6	5	4	3	2	1	0	
0	1	0	0	0	0	1	0	(5000H)

RST p

Operation: $(SP-1) \leftarrow PC_H$, $(SP-2) \leftarrow PC_L$, $PC_H \leftarrow 0$, $PC_L \leftarrow P$

Format:

Opcode	Operand
RST	p



Description:

The current Program Counter (PC) contents are pushed onto the external memory stack, and the page zero memory location given by operand p is loaded into the PC. Program execution then begins with the opcode in the address now pointed to by PC. The push is performed by first decrementing the contents of the Stack Pointer (SP), loading the high-order byte of PC into the memory address now pointed to by SP, decrementing SP again, and loading the low-order byte of PC into the address now pointed to by SP. The ReSTart instruction allows for a jump to one of eight addresses as shown in the table below. The operand p is assembled into the object code using the corresponding T state. Note: Since all addresses are in page zero of memory, the high order byte of PC is loaded with 00H. The number selected from the "p" column of the table is loaded into the low-order byte of PC.

p	t
00H	000
08H	001
10H	010
18H	011
20H	100
28H	101
30H	110
38H	111

M CYCLES: 3 T STATES: 11(5,3,3)

Example:

If the contents of the Program Counter are 15B3H, after the execution of

RST 18H (Object code 1101111)

the PC will contain 0018H, as the address of the next opcode to be fetched.

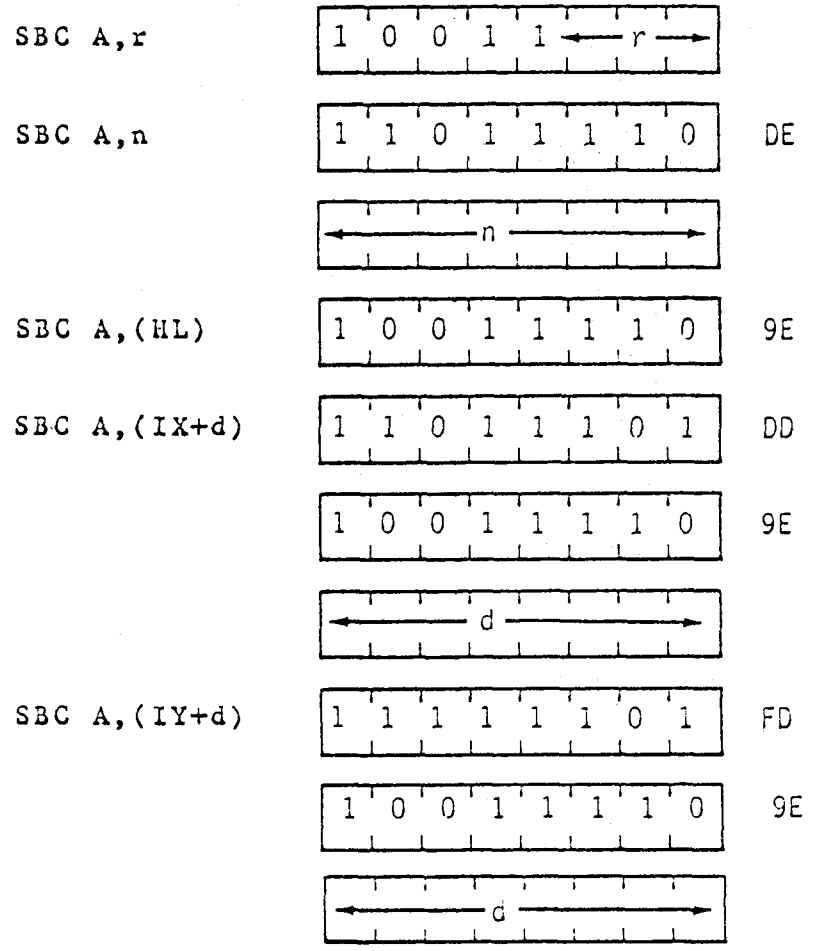
SBC A, s

Operation: A ← A - s - CY

Format:

Opcode	Operands
SBC	A,s

The s operand is any of r,n,(HL),(IX+d) or (IY+d) as defined for the analogous ADD instructions. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B,C,D,E,H,L or A assembled as follows in the object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

The s operand, along with the Carry Flag ("C" in the F register) is subtracted from the contents of the Accumulator, and the result is stored in the Accumulator.

INSTRUCTION	M CYCLES	T STATES
SBC A,r	1	4
SBC A,n	2	7(4,3)
SBC A,(HL)	2	7(4,3)
SBC A,(IX+d)	5	19(4,4,3,5,3)
SBC A,(IY+d)	5	19(4,4,3,5,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if borrow from Bit 4; reset otherwise
P/V:	Set if overflow; reset otherwise
N:	Set
C:	Set if borrow; reset otherwise

Example:

If the Accumulator contains 16H, the carry flag is set, the HL register pair contains 3433H, and address 3433H contains 05H, after the execution of

```
SBC A,(HL)
```

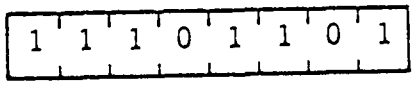
the Accumulator will contain 10H.

SBC HL, ss

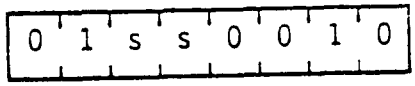
Operation: HL←HL-ss-CY

Format:

Opcode	Operands
SBC	HL,ss



ED



Description:

The contents of the register pair ss (any of register pairs BC,DE,HL or SP) and the Carry Flag (C flag in the F register) are subtracted from the contents of register pair HL and the result is stored in HL. Operand ss is specified as follows in the assembled object code.

Register Pair	ss
BC	00
DE	00
HL	10
SP	11

M CYCLES: 4 T STATES: 15(4,4,4,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if borrow from Bit 12; reset otherwise
P/V:	Set if overflow; reset otherwise
N:	Set
C:	Set if borrow; reset otherwise

Example:

If the contents of the HL register pair are 9999H, the contents of register pair DE are 1111H, and the Carry Flag is set, after the execution of

```
SBC HL,DE
```

the contents of HL will be 8887H.

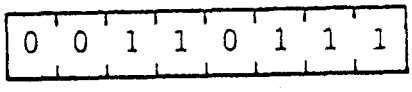
SCF

Operation: CY ← 1

Format:

Opcode

SCF



37

Description:

The C flag in the F register is set.

M CYCLES: 1 T STATES: 4

Condition Bits Affected:

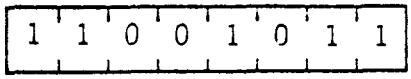
S:	Not affected
Z:	Not affected
H:	Reset
P/V:	Not affected
N:	Reset
C:	Set

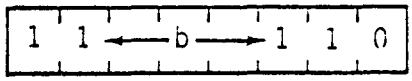
SET b, (HL)

Operation: $(HL)_b \leftarrow 1$

Format:

Opcode	Operands
SET	b,(HL)





Description:

Bit b (any bit, 7 through 0) in the memory location addressed by the contents of register pair HL is set. Operand b is specified as follows in the assembled object code:

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

M CYCLES: 4 T STATES: 15(4,4,4,3)

Condition Bits Affected: None

Example:

If the contents of the HL register pair are 3000H, after the execution of

```
SET 4,(HL)
```

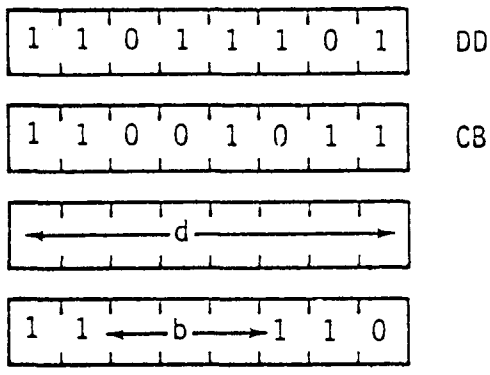
bit 4 in memory location 3000H will be 1. (Bit 0 in memory location 3000H is the least significant bit.)

SET b, (IX+d)

Operation: $(IX+d)_b \leftarrow 1$

Format:

Opcode	Operands
SET	b,(IX+d)



Description:

Bit b (any bit, 7 through 0) in the memory location addressed by the sum of the contents of the IX register pair (Index Register IX) and the two's complement integer d is set. Operand b is specified as follows in the assembled object code:

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

M CYCLES: 6 T STATES: 23(4,4,3,5,4,3)

Condition Bits Affected: None

Example:

If the contents of Index Register are 2000H, after the execution of

```
SET 0,(IX+3H)
```

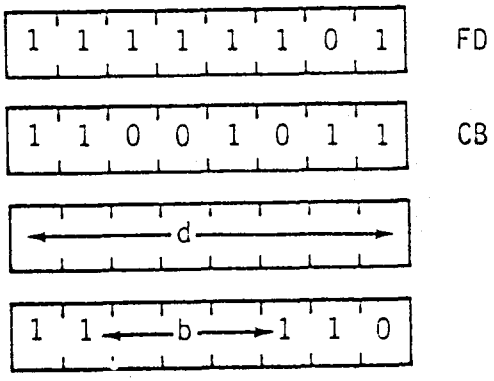
bit 0 in memory location 2003H will be 1. (Bit 0 in memory location 2003H is the least significant bit.)

SET b, (IY+d)

Operation: $(IY+d)_b \leftarrow 1$

Format:

Opcode	Operands
SET	b,(IY+d)



Description:

Bit b (any bit, 7 through 0) in the memory location addressed by the sum of the contents of the IY register pair (Index Register IY) and the two's complement displacement d is set. Operand b is specified as follows in the assembled object code:

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

M CYCLES: 6 T STATES: 23(4,4,3,5,4,3)

Condition Bits Affected: None

Example:

If the contents of Index Register IY are 2000H, after the execution of

```
SET 0,(IY+3H)
```

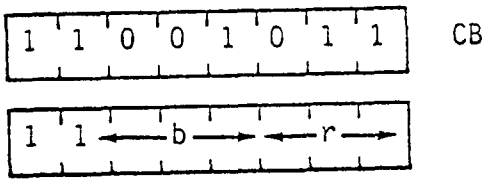
bit 0 in memory location 2003H will be 1. (Bit 0 in memory location 2003H is the least significant bit.)

SET b, r

Operation: $r_b \leftarrow 1$

Format:

Opcode	Operands
SET	b,r



Description:

Bit b (any bit, 7 through 0) in register r (any of registers B,C,D,E,H,L or A) is set. Operands b and r are specified as follows in the assembled object code:

Bit	b	Register	r
0	000	B	000
1	001	C	001
2	010	D	010
3	011	E	011
4	100	H	100
5	101	L	101
6	110	A	111
7	111		

M CYCLES: 2 T STATES: 8(4,4)

Condition Bits Affected: None

Example:

After the execution of

SET 4,A

bit 4 in register A will be set. (Bit 0 is the least significant bit.)

SLA m

Operation: m CY ← 7 ← 0 ← 0

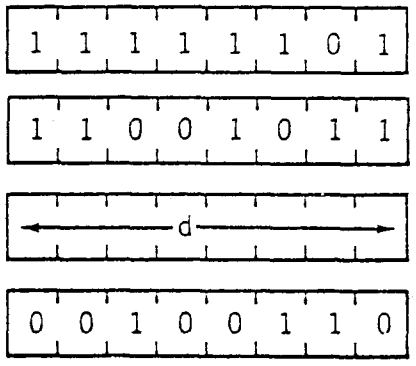
Format:

Opcode	Operands
SLA	m

The operand m is any of r, (HL), (IX+d) or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:

SLA r	1	1	0	0	1	0	1	1	CB
	0	0	1	0	0	←	r	→	
SLA (HL)	1	1	0	0	1	0	1	1	CB
	0	0	1	0	0	1	1	0	26
SLA (IX+d)	1	1	0	1	1	1	0	1	DD
	1	1	0	0	1	0	1	1	CB
				d					
	0	0	1	0	0	1	1	0	26

SLA (IY+d)



*r identifies registers B,C,D,E,H,L or A specified as follows in the assembled object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

An arithmetic shift left is performed on the contents of operand m: bit 0 is reset, the previous content of bit 0 is copied into bit 1, the previous content of bit 1 is copied into bit 2; this pattern is continued throughout; the content of bit 7 is copied into the Carry Flag (C flag in register F). Bit 0 is the least significant bit.

INSTRUCTION	M CYCLES	T STATES
SLA r	2	8(4,4)
SLA (HL)	4	15(4,4,4,3)
SLA (IX+d)	6	23(4,4,3,5,4,3)
SLA (IY+d)	6	23(4,4,3,5,4,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Reset
P/V:	Set if parity is even; reset otherwise
N:	Reset
C:	Data from Bit 7

Example:

If the contents of register L are

7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	1

after the execution of

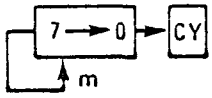
SLA L

the contents of register L and the Carry Flag will be

C	7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	1	0

SRA m

Operation:



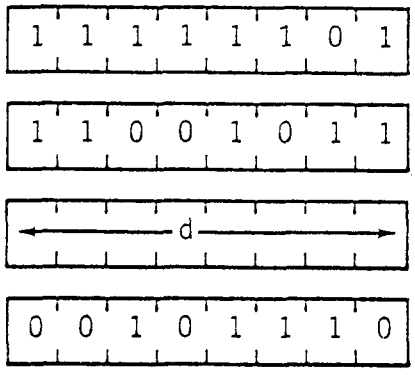
Format:

Opcode	Operands
SRA	m

The m operand is any of r, (HL), (IX+d) or (IY+d),as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:

SRA r	1 1 0 0 1 0 1 1	CB
	0 0 1 0 1 ←— r —→	
SRA(HL)	1 1 0 0 1 0 1 1	CB
	0 0 1 0 1 1 1 0	2E
SRA(IX+d)	1 1 0 1 1 1 0 1	DD
	1 1 0 0 1 0 1 1	CB
	←——— d ———→	
	0 0 1 0 1 1 1 0	2E

SRA(IY+d)



*r means registers B,C,D,E,H,L or A specified as follows in the assembled object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

An arithmetic shift right is performed on the contents of operand m: the content of bit 7 is copied into bit 6; the previous content of bit 6 is copied into bit 5; this pattern is continued throughout the byte. The content of bit 0 is copied into the Carry Flag (C flag in register F), and the previous content of bit 7 is unchanged. Bit 0 is the least significant bit.

INSTRUCTION	M CYCLES	T STATES
SRA r	2	8(4,4)
SRA (HL)	4	15(4,4,4,3)
SRA (IX+d)	6	23(4,4,3,5,4,3)
SRA (IY+d)	6	23(4,4,3,5,4,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Reset
P/V:	Set if parity is even; reset otherwise
N:	Reset
C:	Data from Bit 0 of source register

Example:

If the contents of the Index Register IX are 1000H, and the contents of memory location 1003H are

7	6	5	4	3	2	1	0
1	0	1	1	1	0	0	0

after the execution of

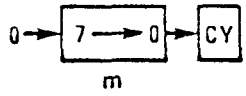
SRA (IX+3H)

the contents of memory location 1003H and the Carry Flag will be

7	6	5	4	3	2	1	0	C
1	1	0	1	1	1	0	0	0

SRL m

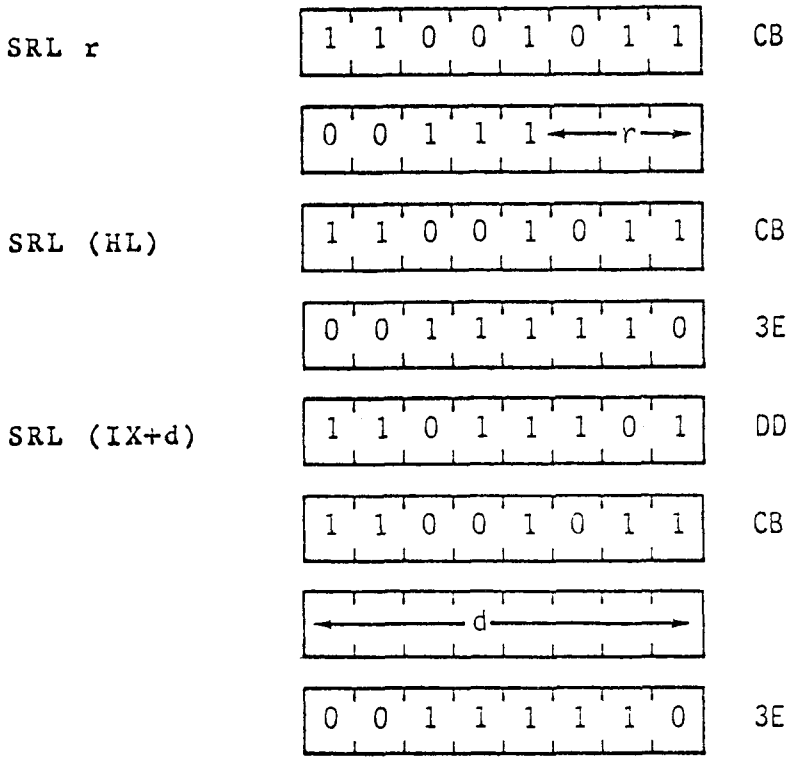
Operation:



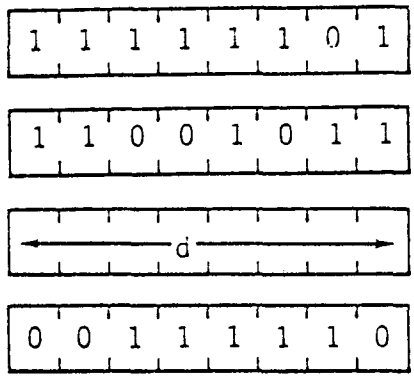
Format:

Opcode	Operands
SRL	m

The operand m is any of r, (HL), (IX+d) or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:



SRL (IY+d)



*r identifies registers B,C,D,E,H,L or A specified as follows in the assembled object code fields above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

The contents of operand m are shifted right: the content of bit 7 is copied into bit 6; the content of bit 6 is copied into bit 5; this pattern is continued throughout the byte. The content of bit 0 is copied into the Carry Flag, and bit 7 is reset. Bit 0 is the least significant bit.

INSTRUCTION	M CYCLES	T STATES
SRL r	2	8(4,4)
SRL (HL)	4	15(4,4,4,3)
SRL (IX+d)	6	23(4,4,3,5,4,3)
SRL (IY+d)	6	23(4,4,3,5,4,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Reset
P/V:	Set if parity is even; reset otherwise
N:	Reset
C:	Data from Bit 0 of source register

Example:

If the contents of register B are

7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	1

after the execution of

```
SRL B
```

the contents of register B and the Carry Flag will be

/	6	5	4	3	2	1	0	c
0	1	0	0	0	1	1	1	1

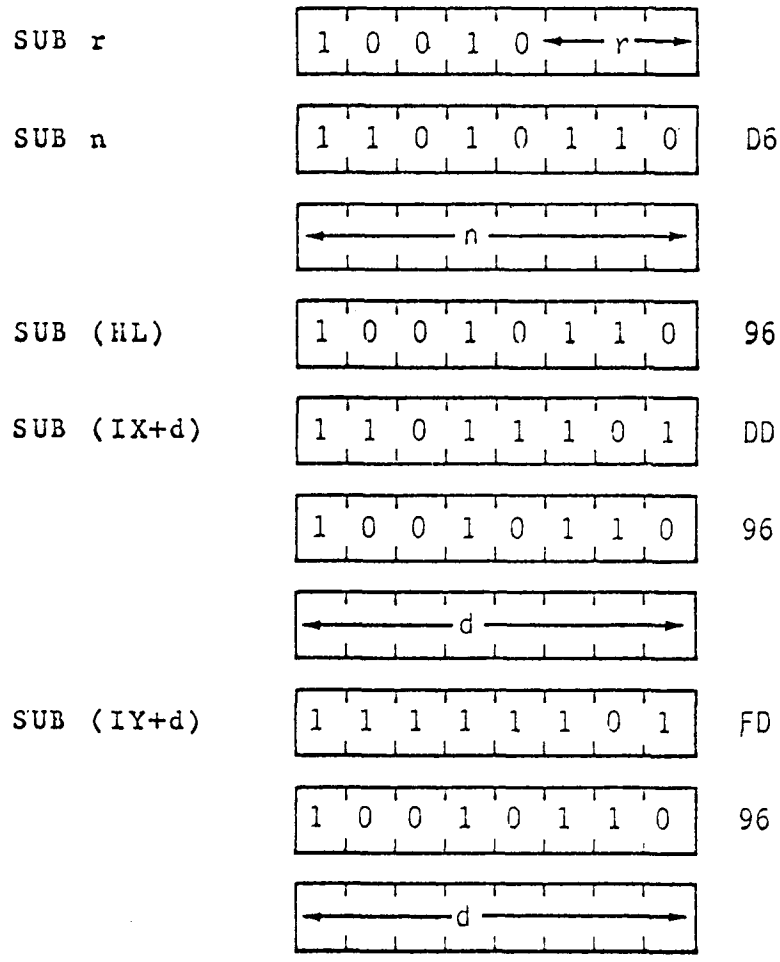
SUB s

Operation: A ← A - s

Format:

Opcode	Operands
SUB	s

The s operand is any of r,n,(HL),(IX+d) or (IY+d) as defined for the analogous ADD instruction. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B,C,D,E,H,L or A assembled as follows in the object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

The s operand is subtracted from the contents of the Accumulator, and the result is stored in the Accumulator.

INSTRUCTION	M CYCLES	T STATES
SUB r	1	4
SUB n	2	7(4,3)
SUB (HL)	2	7(4,3)
SUB (IX+d)	5	19(4,4,3,5,3)
SUB (IY+d)	5	19(4,4,3,5,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if borrow from Bit 4; reset otherwise
P/V:	Set if overflow; reset otherwise
N:	Set
C:	Set if borrow; reset otherwise

Example:

If the Accumulator contains 29H and register D contains 11H, after the execution of

```
SUB D
```

the Accumulator will contain 18H.

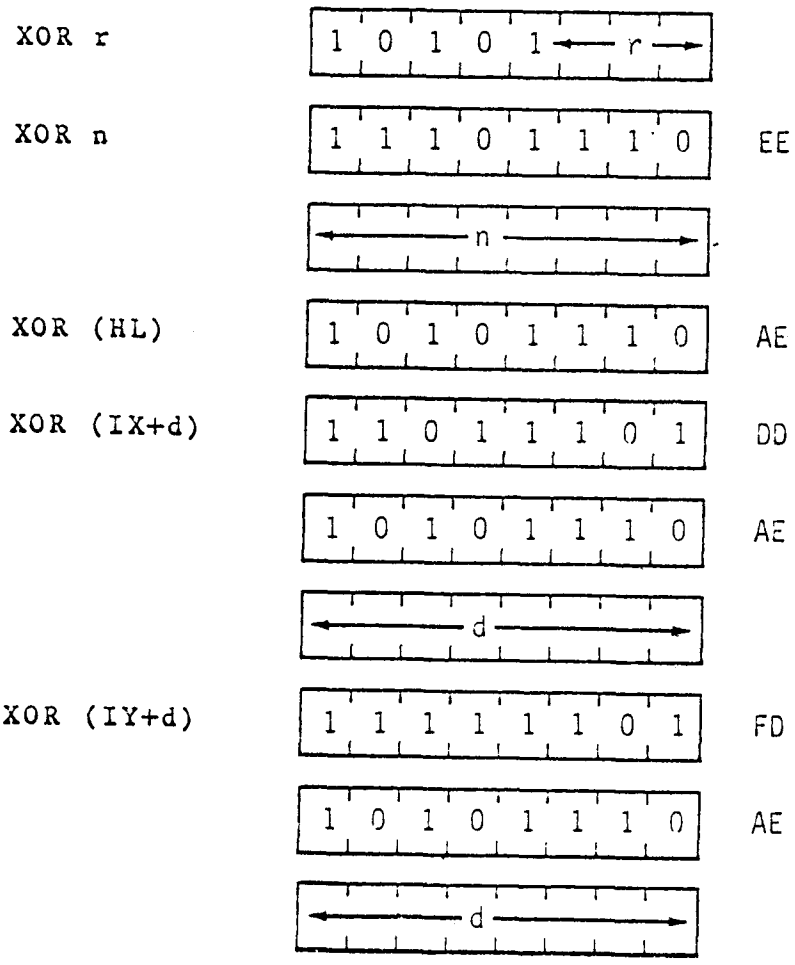
XOR s

Operation: $A \leftarrow A \oplus s$

Format:

Opcode	Operands
XOR	s

The s operand is any of r,n, (HL),(IX+d) or (IY+d), as defined for the analogous ADD instructions. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B,C,D,E,H,L or A assembled as follows in the object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

A logical exclusive-OR operation, bit by bit, is performed between the byte specified by the s operand and the byte contained in the Accumulator; the result is stored in the Accumulator.

INSTRUCTION	M CYCLES	T STATES
XOR r	1	4
XOR n	2	7(4,3)
XOR (HL)	2	7(4,3)
XOR (IX+d)	5	19(4,4,3,5,3)
XOR (IY+d)	5	19(4,4,3,5,3)

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set
P/V:	Set if parity even; reset otherwise
N:	Reset
C:	Reset

Example:

If the Accumulator contains 96H (10010110), after the execution of

XOR 5DH (Note: 5DH = 01011101)

the Accumulator will contain CBH (11001011).

APPENDIX A

ALPHABETICAL LISTING OF Z80 OPCODES

```
 LOC   OBJ CODE    STMT SOURCE STATEMENT

                   0001 *H APPENDIX 1
0000   8E          0002           ADC   A,(HL)
0001   DD8E05      0003           ADC   A,(IX+IND)
0004   FD8E05      0004           ADC   A,(IY+IND)
0007   8F          0005           ADC   A,A
0008   88          0006           ADC   A,B
0009   89          0007           ADC   A,C
000A   8A          0008           ADC   A,D
000B   8B          0009           ADC   A,E
000C   8C          0010           ADC   A,H
000D   8D          0011           ADC   A,L
000E   CE20        0012           ADC   A,N
0010   ED4A        0013           ADC   HL,BC
0012   ED5A        0014           ADC   HL,DE
0014   ED6A        0015           ADC   HL,HL
0016   ED7A        0016           ADC   HL,SP
                   0017 ;
0018   86          0018           ADD   A,(HL)
0019   DD8605      0019           ADD   A,(IX+IND)
001C   FD8605      0020           ADD   A,(IY+IND)
001F   87          0021           ADD   A,A
0020   80          0022           ADD   A,B
0021   81          0023           ADD   A,C
0022   82          0024           ADD   A,D
0023   83          0025           ADD   A,E
0024   84          0026           ADD   A,H
0025   85          0027           ADD   A,L
0026   C620        0028           ADD   A,N
0028   09          0029           ADD   HL,BC
0029   19          0030           ADD   HL,DE
002A   29          0031           ADD   HL,HL
002B   39          0032           ADD   HL,SP
002C   DD09        0033           ADD   IX,BC
002E   DD19        0034           ADD   IX,DE
0030   DD29        0035           ADD   IX,IX
0032   DD39        0036           ADD   IX,SP
0034   FD09        0037           ADD   IY,BC
0036   FD19        0038           ADD   IY,DE
0038   FD29        0039           ADD   IY,IY
003A   FD39        0040           ADD   IY,SP
                   0041 ;
003C   A6          0042           AND   (HL)
003D   DDA605      0043           AND   (IX+IND)
0040   FDA605      0044           AND   (IY+IND)
0043   A7          0045           AND   A
0044   A0          0046           AND   B
0045   A1          0047           AND   C
0046   A2          0048           AND   D
0047   A3          0049           AND   E
0048   A4          0050           AND   H
0049   A5          0051           AND   L
004A   E620        0052           AND   N
                   0053 ;
004C   CB46        0054           BIT   0,(HL)
004E   DDCB0546    0055           BIT   0,(IX+IND)
0052   FDCB0546    0056           BIT   0,(IY+IND)
0056   CB47        0057           BIT   0,A
0058   CB40        0058           BIT   0,B
```

```
 LOC    OBJ CODE    STMT SOURCE STATEMENT

005A    CB41        0059            BIT   0,C
005C    CB42        0060            BIT   0,D
005E    CB43        0061            BIT   0,E
0060    CB44        0062            BIT   0,H
0062    CB45        0063            BIT   0,L
0064    CB4E        0064            BIT   1,(HL)
0066    DDCB054E    0065            BIT   1,(IX+IND)
006A    FDCB054E    0066            BIT   1,(IY+IND)
006E    CB4F        0067            BIT   1,A
0070    CB48        0068            BIT   1,B
0072    CB49        0069            BIT   1,C
0074    CB4A        0070            BIT   1,D
0076    CB4B        0071            BIT   1,E
0078    CB4C        0072            BIT   1,H
007A    CB4D        0073            BIT   1,L
                    0074 ;
007C    CB56        0075            BIT   2,(HL)
007E    DDCB0556    0076            BIT   2,(IX+IND)
0082    FDCB0556    0077            BIT   2,(IY+IND)
0086    CB57        0078            BIT   2,A
0088    CB50        0079            BIT   2,B
008A    CB51        0080            BIT   2,C
008C    CB52        0081            BIT   2,D
008E    CB53        0082            BIT   2,E
0090    CB54        0083            BIT   2,H
0092    CB55        0084            BIT   2,L
                    0085 ;
0094    CB5E        0086            BIT   3,(HL)
0096    DDCB055E    0087            BIT   3,(IX+IND)
009A    FDCB055E    0088            BIT   3,(IY+IND)
009E    CB5F        0089            BIT   3,A
00A0    CB58        0090            BIT   3,B
00A2    CB59        0091            BIT   3,C
00A4    CB5A        0092            BIT   3,D
00A6    CB5B        0093            BIT   3,E
00A8    CB5C        0094            BIT   3,H
00AA    CB5D        0095            BIT   3,L
                    0096 ;
00AC    CB66        0097            BIT   4,(HL)
00AE    DDCB0566    0098            BIT   4,(IX+IND)
00B2    FDCB0566    0099            BIT   4,(IY+IND)
00B6    CB67        0100            BIT   4,A
00B8    CB60        0101            BIT   4,B
00BA    CB61        0102            BIT   4,C
00BC    CB62        0103            BIT   4,D
00BE    CB63        0104            BIT   4,E
00C0    CB64        0105            BIT   4,H
00C2    CB65        0106            BIT   4,L
                    0107 ;
00C4    CB6E        0108            BIT   5,(HL)
00C6    DDCB056E    0109            BIT   5,(IX+IND)
00CA    FDCB056E    0110            BIT   5,(IY+IND)
00CE    CB6F        0111            BIT   5,A
00D0    CB68        0112            BIT   5,B
00D2    CB69        0113            BIT   5,C
00D4    CB6A        0114            BIT   5,D
00D6    CB6B        0115            BIT   5,E
00D8    CB6C        0116            BIT   5,H
```

```
 LOC    OBJ CODE    STMT SOURCE STATEMENT

00DA    CB6D        0117            BIT   5,L
                    0118 ;
00DC    CB76        0119            BIT   6,(HL)
00DE    DDCB0576    0120            BIT   6,(IX+IND)
00E2    FDCB0576    0121            BIT   6,(IY+IND)
00E6    CB77        0122            BIT   6,A
00E8    CB70        0123            BIT   6,B
00EA    CB71        0124            BIT   6,C
00EC    CB72        0125            BIT   6,D
00EE    CB73        0126            BIT   6,E
00F0    CB74        0127            BIT   6,H
00F2    CB75        0128            BIT   6,L
                    0129 ;
00F4    CB7E        0130            BIT   7,(HL)
00F6    DDCB057E    0131            BIT   7,(IX+IND)
00FA    FDCB057E    0132            BIT   7,(IY+IND)
00FE    CB7F        0133            BIT   7,A
0100    CB78        0134            BIT   7,B
0102    CB79        0135            BIT   7,C
0104    CB7A        0136            BIT   7,D
0106    CB7B        0137            BIT   7,E
0108    CB7C        0138            BIT   7,H
010A    CB7D        0139            BIT   7,L
                    0140 ;
010C    DC8805      0141            CALL  C,NN
010F    FC8805      0142            CALL  M,NN
0112    D48805      0143            CALL  NC,NN
0115    CD8805      0144            CALL  NN
0118    C48805      0145            CALL  NZ,NN
011B    F48805      0146            CALL  P,NN
011E    EC8805      0147            CALL  PE,NN
0121    E48805      0148            CALL  PO,NN
0124    CC8805      0149            CALL  Z,NN
                    0150 ;
0127    3F          0151            CCF
                    0152 ;
0128    BE          0153            CP    (HL)
0129    DDBE05      0154            CP    (IX+IND)
012C    FDBE05      0155            CP    (IY+IND)
012F    BF          0156            CP    A
0130    B8          0157            CP    B
0131    B9          0158            CP    C
0132    BA          0159            CP    D
0133    BB          0160            CP    E
0134    BC          0161            CP    H
0135    BD          0162            CP    L
0136    FE20        0163            CP    N
                    0164 ;
0138    EDA9        0165            CPD
013A    EDB9        0166            CPDR
013C    EDA1        0167            CPI
013E    EDB1        0168            CPIR
                    0169 ;
0140    2F          0170            CPL
                    0171 ;
0141    27          0172            DAA
                    0173 ;
0142    35          0174            DEC   (HL)
```

```
 LOC    OBJ CODE    STMT SOURCE STATEMENT

0143    DD3505      0175            DEC     (IX+IND)
0146    FD3505      0176            DEC     (IY+IND)
0149    3D          0177            DEC     A
014A    05          0178            DEC     B
014B    0B          0179            DEC     BC
014C    0D          0180            DEC     C
014D    15          0181            DEC     D
014E    1B          0182            DEC     DE
014F    1D          0183            DEC     E
0150    25          0184            DEC     H
0151    2B          0185            DEC     HL
0152    DD2B        0186            DEC     IX
0154    FD2B        0187            DEC     IY
0156    2D          0188            DEC     L
0157    3B          0189            DEC     SP
                    0190 ;
0158    F3          0191            DI
                    0192 ;
0159    102E        0193            DJNZ    DIS
                    0194 ;
015B    FB          0195            EI
                    0196 ;
015C    E3          0197            EX      (SP),HL
015D    DDE3        0198            EX      (SP),IX
015F    FDE3        0199            EX      (SP),IY
0161    08          0200            EX      AF,AF'
0162    EB          0201            EX      DE,HL
0163    D9          0202            EXX
                    0203 ;
0164    76          0204            HALT
                    0205 ;
0165    ED46        0206            IM      0
0167    ED56        0207            IM      1
0169    ED5E        0208            IM      2
                    0209 ;
016B    ED78        0210            IN      A,(C)
016D    DB20        0211            IN      A,(N)
016F    ED40        0212            IN      B,(C)
0171    ED48        0213            IN      C,(C)
0173    ED50        0214            IN      D,(C)
0175    ED58        0215            IN      E,(C)
0177    ED60        0216            IN      H,(C)
0179    ED68        0217            IN      L,(C)
                    0218 ;
017B    34          0219            INC     (HL)
017C    DD3405      0220            INC     (IX+IND)
017F    FD3405      0221            INC     (IY+IND)
0182    3C          0222            INC     A
0183    04          0223            INC     B
0184    03          0224            INC     BC
0185    0C          0225            INC     C
0186    14          0226            INC     D
0187    13          0227            INC     DE
0188    1C          0228            INC     E
0189    24          0229            INC     H
018A    23          0230            INC     HL
018B    DD23        0231            INC     IX
018D    FD23        0232            INC     IY
```

```
 LOC   OBJ CODE    STMT SOURCE STATEMENT

018F   2C          0233          INC   L
0190   33          0234          INC   SP
                   0235 ;
0191   EDAA        0236          IND
0193   EDBA        0237          INDR
0195   EDA2        0238          INI
0197   EDB2        0239          INIR
                   0240 ;
0199   E9          0241          JP    (HL)
019A   DDE9        0242          JP    (IX)
019C   FDE9        0243          JP    (IY)
019E   DA8805      0244          JP    C,NN
01A1   FA8805      0245          JP    M,NN
01A4   D28805      0246          JP    NC,NN
01A7   C38805      0247          JP    NN
01AA   C28805      0248          JP    NZ,NN
01AD   F28805      0249          JP    P,NN
01B0   EA8805      0250          JP    PE,NN
01B3   E28805      0251          JP    PO,NN
01B6   CA8805      0252          JP    Z,NN
                   0253 ;
01B9   382E        0254          JR    C,DIS
01BB   182E        0255          JR    DIS
01BD   302E        0256          JR    NC,DIS
01BF   202E        0257          JR    NZ,DIS
01C1   282E        0258          JR    Z,DIS
                   0259 ;
01C3   02          0260          LD    (BC),A
01C4   12          0261          LD    (DE),A
01C5   77          0262          LD    (HL),A
01C6   70          0263          LD    (HL),B
01C7   71          0264          LD    (HL),C
01C8   72          0265          LD    (HL),D
01C9   73          0266          LD    (HL),E
01CA   74          0267          LD    (HL),H
01CB   75          0268          LD    (HL),L
01CC   3620        0269          LD    (HL),N
01CE   DD7705      0270          LD    (IX+IND),A
01D1   DD7005      0271          LD    (IX+IND),B
01D4   DD7105      0272          LD    (IX+IND),C
01D7   DD7205      0273          LD    (IX+IND),D
01DA   DD7305      0274          LD    (IX+IND),E
01DD   DD7405      0275          LD    (IX+IND),H
01E0   DD7505      0276          LD    (IX+IND),L
01E3   DD360520    0277          LD    (IX+IND),N
                   0278 ;
01E7   FD7705      0279          LD    (IY+IND),A
01EA   FD7005      0280          LD    (IY+IND),B
01ED   FD7105      0281          LD    (IY+IND),C
01F0   FD7205      0282          LD    (IY+IND),D
01F3   FD7305      0283          LD    (IY+IND),E
01F6   FD7405      0284          LD    (IY+IND),H
01F9   FD7505      0285          LD    (IY+IND),L
01FC   FD360520    0286          LD    (IY+IND),N
                   0287 ;
0200   328805      0288          LD    (NN),A
0203   ED438805    0289          LD    (NN),BC
0207   ED538805    0290          LD    (NN),DE
```

```
 LOC    OBJ CODE    STMT SOURCE STATEMENT

0208    228805      0291            LD      (NN),HL
020E    DD228805    0292            LD      (NN),IX
0212    FD228805    0293            LD      (NN),IY
0216    ED738305    0294            LD      (NN),SP
                    0295 ;
021A    0A          0296            LD      A,(BC)
021B    1A          0297            LD      A,(DE)
021C    7E          0298            LD      A,(HL)
021D    DD7E05      0299            LD      A,(IX+IND)
0220    FD7E05      0300            LD      A,(IY+IND)
0223    3A8805      0301            LD      A,(NN)
0226    7F          0302            LD      A,A
0227    78          0303            LD      A,B
0228    79          0304            LD      A,C
0229    7A          0305            LD      A,D
022A    7B          0306            LD      A,E
022B    7C          0307            LD      A,H
022C    ED57        0308            LD      A,I
022E    7D          0309            LD      A,L
022F    3E20        0310            LD      A,N
0231    ED5F        0311            LD      A,R
                    0312 ;
0233    46          0313            LD      B,(HL)
0234    DD4605      0314            LD      B,(IX+IND)
0237    FD4605      0315            LD      B,(IY+IND)
023A    47          0316            LD      B,A
023B    40          0317            LD      B,B
023C    41          0318            LD      B,C
023D    42          0319            LD      B,D
023E    43          0320            LD      B,E
023F    44          0321            LD      B,H
0240    45          0322            LD      B,L
0241    0620        0323            LD      B,N
                    0324 ;
0243    ED4B8805    0325            LD      BC,(NN)
0247    018805      0326            LD      BC,NN
                    0327 ;
024A    4E          0328            LD      C,(HL)
024B    DD4E05      0329            LD      C,(IX+IND)
024E    FD4E05      0330            LD      C,(IY+IND)
0251    4F          0331            LD      C,A
0252    48          0332            LD      C,B
0253    49          0333            LD      C,C
0254    4A          0334            LD      C,D
0255    4B          0335            LD      C,E
0256    4C          0336            LD      C,H
0257    4D          0337            LD      C,L
0258    0E20        0338            LD      C,N
                    0339 ;
025A    56          0340            LD      D,(HL)
025B    DD5605      0341            LD      D,(IX+IND)
025E    FD5605      0342            LD      D,(IY+IND)
0261    57          0343            LD      D,A
0262    50          0344            LD      D,B
0263    51          0345            LD      D,C
0264    52          0346            LD      D,D
0265    53          0347            LD      D,E
0266    54          0348            LD      D,H
```

```
 LOC    OBJ CODE    STMT SOURCE STATEMENT

0267    55          0349            LD      D,L
0268    1620        0350            LD      D,N
                    0351 ;
026A    ED5B8805    0352            LD      DE,(NN)
026E    118805      0353            LD      DE,NN
                    0354 ;
0271    5E          0355            LD      E,(HL)
0272    DD5E05      0356            LD      E,(IX+IND)
0275    FD5E05      0357            LD      E,(IY+IND)
0278    5F          0358            LD      E,A
0279    58          0359            LD      E,B
027A    59          0360            LD      E,C
027B    5A          0361            LD      E,D
027C    5B          0362            LD      E,E
027D    5C          0363            LD      E,H
027E    5D          0364            LD      E,L
027F    1E20        0365            LD      E,N
                    0366 ;
0281    66          0367            LD      H,(HL)
0282    DD6605      0368            LD      H,(IX+IND)
0285    FD6605      0369            LD      H,(IY+IND)
0288    67          0370            LD      H,A
0289    60          0371            LD      H,B
028A    61          0372            LD      H,C
028B    62          0373            LD      H,D
028C    63          0374            LD      H,E
028D    64          0375            LD      H,H
028E    65          0376            LD      H,L
028F    2620        0377            LD      H,N
                    0378 ;
0291    2A8805      0379            LD      HL,(NN)
0294    218805      0380            LD      HL,NN
                    0381 ;
0297    ED47        0382            LD      I,A
                    0383 ;
0299    DD2A8805    0384            LD      IX,(NN)
029D    DD218805    0385            LD      IX,NN
                    0386 ;
02A1    FD2A8805    0387            LD      IY,(NN)
02A5    FD218805    0388            LD      IY,NN
                    0389 ;
02A9    6E          0390            LD      L,(HL)
02AA    DD6E05      0391            LD      L,(IX+IND)
02AD    FD6E05      0392            LD      L,(IY+IND)
02B0    6F          0393            LD      L,A
02B1    68          0394            LD      L,B
02B2    69          0395            LD      L,C
02B3    6A          0396            LD      L,D
02B4    6B          0397            LD      L,E
02B5    6C          0398            LD      L,H
02B6    6D          0399            LD      L,L
02B7    2E20        0400            LD      L,N
                    0401 ;
02B9    ED4F        0402            LD      R,A
                    0403 ;
02BB    ED7B8805    0404            LD      SP,(NN)
02BF    F9          0405            LD      SP,HL
02C0    DDF9        0406            LD      SP,IX
```

```
 LOC   OBJ CODE    STMT SOURCE STATEMENT

02C2   FDF9        0407            LD      SP,IY
02C4   318803      0408            LD      SP,NN
                   0409 ;
02C7   EDA8        0410            LDD
02C9   EDB8        0411            LDDR
02CB   EDA0        0412            LDI
02CD   EDB0        0413            LDIR
                   0414 ;
02CF   ED44        0415            NEG
                   0416 ;
02D1   00          0417            NOP
                   0418 ;
02D2   B6          0419            OR      (HL)
02D3   DDB605      0420            OR      (IX+IND)
02D6   FDB605      0421            OR      (IY+IND)
02D9   B7          0422            OR      A
02DA   B0          0423            OR      B
02DB   B1          0424            OR      C
02DC   B2          0425            OR      D
02DD   B3          0426            OR      E
02DE   B4          0427            OR      H
02DF   B5          0428            OR      L
02E0   F620        0429            OR      N
                   0430 ;
02E2   EDBB        0431            OTDR
02E4   EDB3        0432            OTIR
                   0433 ;
02E6   ED79        0434            OUT     (C),A
02E8   ED41        0435            OUT     (C),B
02EA   ED49        0436            OUT     (C),C
02EC   ED51        0437            OUT     (C),D
02EE   ED59        0438            OUT     (C),E
02F0   ED61        0439            OUT     (C),H
02F2   ED69        0440            OUT     (C),L
02F4   D320        0441            OUT     (N),A
                   0442 ;
02F6   EDAB        0443            OUTD
02F8   EDA3        0444            OUTI
                   0445 ;
02FA   F1          0446            POP     AF
02FB   C1          0447            POP     BC
02FC   D1          0448            POP     DE
02FD   E1          0449            POP     HL
02FE   DDE1        0450            POP     IX
0300   FDE1        0451            POP     IY
0302   F5          0452            PUSH    AF
0303   C5          0453            PUSH    BC
0304   D5          0454            PUSH    DE
0305   E5          0455            PUSH    HL
0306   DDE5        0456            PUSH    IX
0308   FDE5        0457            PUSH    IY
                   0458 ;
030A   CB86        0459            RES     0,(HL)
030C   DDCB0586    0460            RES     0,(IX+IND)
0310   FDCB0586    0461            RES     0,(IY+IND)
0314   CB87        0462            RES     0,A
0316   CB80        0463            RES     0,B
0318   CB81        0464            RES     0,C
```

```
 LOC    OBJ CODE    STMT SOURCE STATEMENT

031A    CB82        0465            RES   0,D
031C    CB83        0466            RES   0,E
031E    CB84        0467            RES   0,H
0320    CB85        0468            RES   0,L
                    0469 ;
0322    CB8E        0470            RES   1,(HL)
0324    DDCB058E    0471            RES   1,(IX+IND)
0328    FDCB058E    0472            RES   1,(IY+IND)
032C    CB8F        0473            RES   1,A
032E    CB88        0474            RES   1,B
0330    CB89        0475            RES   1,C
0332    CB8A        0476            RES   1,D
0334    CB8B        0477            RES   1,E
0336    CB8C        0478            RES   1,H
0338    CB8D        0479            RES   1,L
                    0480 ;
033A    CB96        0481            RES   2,(HL)
033C    DDCB0596    0482            RES   2,(IX+IND)
0340    FDCB0596    0483            RES   2,(IY+IND)
0344    CB97        0484            RES   2,A
0346    CB90        0485            RES   2,B
0348    CB91        0486            RES   2,C
034A    CB92        0487            RES   2,D
034C    CB93        0488            RES   2,E
034E    CB94        0489            RES   2,H
0350    CB95        0490            RES   2,L
                    0491 ;
0352    CB9E        0492            RES   3,(HL)
0354    DDCB059E    0493            RES   3,(IX+IND)
0358    FDCB059E    0494            RES   3,(IY+IND)
035C    CB9F        0495            RES   3,A
035E    CB98        0496            RES   3,B
0360    CB99        0497            RES   3,C
0362    CB9A        0498            RES   3,D
0364    CB9B        0499            RES   3,E
0366    CB9C        0500            RES   3,H
0368    CB9D        0501            RES   3,L
                    0502 ;
036A    CBA6        0503            RES   4,(HL)
036C    DDCB05A6    0504            RES   4,(IX+IND)
0370    FDCB05A6    0505            RES   4,(IY+IND)
0374    CBA7        0506            RES   4,A
0376    CBA0        0507            RES   4,B
0378    CBA1        0508            RES   4,C
037A    CBA2        0509            RES   4,D
037C    CBA3        0510            RES   4,E
037E    CBA4        0511            RES   4,H
0380    CBA5        0512            RES   4,L
                    0513 ;
0382    CBAE        0514            RES   5,(HL)
0384    DDCB05AE    0515            RES   5,(IX+IND)
0388    FDCB05AE    0516            RES   5,(IY+IND)
038C    CBAF        0517            RES   5,A
038E    CBA8        0518            RES   5,B
0390    CBA9        0519            RES   5,C
0392    CBAA        0520            RES   5,D
0394    CBAB        0521            RES   5,E
0396    CBAC        0522            RES   5,H
```

```
 LOC    OBJ CODE    STMT SOURCE STATEMENT

0398    CBAD        0523            RES   5,L
                    0524 ;
039A    CBB6        0525            RES   6,(HL)
039C    DDCB0536    0526            RES   6,(IX+IND)
03A0    FDCB0536    0527            RES   6,(IY+IND)
03A4    CBB7        0528            RES   6,A
03A6    CBB0        0529            RES   6,B
03A8    CBB1        0530            RES   6,C
03AA    CBB2        0531            RES   6,D
03AC    CBB3        0532            RES   6,E
03AE    CBB4        0533            RES   6,H
03B0    CBB5        0534            RES   6,L
                    0535 ;
03B2    CBBE        0536            RES   7,(HL)
03B4    DDCB053E    0537            RES   7,(IX+IND)
03B8    FDCB053E    0538            RES   7,(IY+IND)
03BC    CBBF        0539            RES   7,A
03BE    CBB8        0540            RES   7,B
03C0    CBB9        0541            RES   7,C
03C2    CBBA        0542            RES   7,D
03C4    CBBB        0543            RES   7,E
03C6    CBBC        0544            RES   7,H
03C8    CBBD        0545            RES   7,L
                    0546 ;
03CA    C9          0547            RET
03CB    D8          0548            RET   C
03CC    F8          0549            RET   M
03CD    D0          0550            RET   NC
03CE    C0          0551            RET   NZ
03CF    F0          0552            RET   P
03D0    E8          0553            RET   PE
03D1    E0          0554            RET   PO
03D2    C8          0555            RET   Z
                    0556 ;
03D3    ED4D        0557            RETI
03D5    ED45        0558            RETN
                    0559 ;
03D7    CB16        0560            RL    (HL)
03D9    DDCB0516    0561            RL    (IX+IND)
03DD    FDCB0516    0562            RL    (IY+IND)
03E1    CB17        0563            RL    A
03E3    CB10        0564            RL    B
03E5    CB11        0565            RL    C
03E7    CB12        0566            RL    D
03E9    CB13        0567            RL    E
03EB    CB14        0568            RL    H
03ED    CB15        0569            RL    L
                    0570 ;
03EF    17          0571            RLA
                    0572 ;
03F0    CB06        0573            RLC   (HL)
03F2    DDCB0506    0574            RLC   (IX+IND)
03F6    FDCB0506    0575            RLC   (IY+IND)
03FA    CB07        0576            RLC   A
03FC    CB00        0577            RLC   B
03FE    CB01        0578            RLC   C
0400    CB02        0579            RLC   D
0402    CB03        0580            RLC   E
```

```
 LOC   OBJ CODE    STMT SOURCE STATEMENT

0404   CB04        0581          RLC   H
0406   CB05        0582          RLC   L
                   0583 ;
0408   07          0584          RLCA
                   0585 ;
0409   ED6F        0586          RLD
                   0587 ;
040B   CB1E        0588          RR    (HL)
040D   DDCB051E    0589          RR    (IX+IND)
0411   FDCB051E    0590          RR    (IY+IND)
0415   CB1F        0591          RR    A
0417   CB18        0592          RR    B
0419   CB19        0593          RR    C
041B   CB1A        0594          RR    D
041D   CB1B        0595          RR    E
041F   CB1C        0596          RR    H
0421   CB1D        0597          RR    L
                   0598 ;
0423   1F          0599          RRA
                   0600 ;
0424   CB0E        0601          RRC   (HL)
0426   DDCB050E    0602          RRC   (IX+IND)
042A   FDCB050E    0603          RRC   (IY+IND)
042E   CB0F        0604          RRC   A
0430   CB08        0605          RRC   B
0432   CB09        0606          RRC   C
0434   CB0A        0607          RRC   D
0436   CB0B        0608          RRC   E
0438   CB0C        0609          RRC   H
043A   CB0D        0610          RRC   L
                   0611 ;
043C   0F          0612          RRCA
                   0613 ;
043D   ED67        0614          RRD
                   0615 ;
043F   C7          0616          RST   0
0440   CF          0617          RST   08H
0441   D7          0618          RST   10H
0442   DF          0619          RST   18H
0443   E7          0620          RST   20H
0444   EF          0621          RST   28H
0445   F7          0622          RST   30H
0446   FF          0623          RST   38H
                   0624 ;
0447   9E          0625          SBC   A,(HL)
0448   DD9E05      0626          SBC   A,(IX+IND)
044B   FD9E05      0627          SBC   A,(IY+IND)
044E   9F          0628          SBC   A,A
044F   98          0629          SBC   A,B
0450   99          0630          SBC   A,C
0451   9A          0631          SBC   A,D
0452   9B          0632          SBC   A,E
0453   9C          0633          SBC   A,H
0454   9D          0634          SBC   A,L
0455   DE20        0635          SBC   A,N
                   0636 ;
0457   ED42        0637          SBC   HL,BC
0459   ED52        0638          SBC   HL,DE
```

```
 LOC    OBJ CODE    STMT SOURCE STATEMENT

045B    ED62        0639            SBC   HL,HL
045D    ED72        0640            SBC   HL,SP
                    0641 ;
045F    37          0642            SCF
                    0643 ;
0460    CBC6        0644            SET   0,(HL)
0462    DDCB05C6    0645            SET   0,(IX+IND)
0466    FDCB05C6    0646            SET   0,(IY+IND)
046A    CBC7        0647            SET   0,A
046C    CBC0        0648            SET   0,B
046E    CBC1        0649            SET   0,C
0470    CBC2        0650            SET   0,D
0472    CBC3        0651            SET   0,E
0474    CBC4        0652            SET   0,H
0476    CBC5        0653            SET   0,L
                    0654 ;
0478    CBCE        0655            SET   1,(HL)
047A    DDCB05CE    0656            SET   1,(IX+IND)
047E    FDCB05CE    0657            SET   1,(IY+IND)
0482    CBCF        0658            SET   1,A
0484    CBC8        0659            SET   1,B
0486    CBC9        0660            SET   1,C
0488    CBCA        0661            SET   1,D
048A    CBCB        0662            SET   1,E
048C    CBCC        0663            SET   1,H
048E    CBCD        0664            SET   1,L
                    0665 ;
0490    CBD6        0666            SET   2,(HL)
0492    DDCB05D6    0667            SET   2,(IX+IND)
0496    FDCB05D6    0668            SET   2,(IY+IND)
049A    CBD7        0669            SET   2,A
049C    CBD0        0670            SET   2,B
049E    CBD1        0671            SET   2,C
04A0    CBD2        0672            SET   2,D
04A2    CBD3        0673            SET   2,E
04A4    CBD4        0674            SET   2,H
04A6    CBD5        0675            SET   2,L
                    0676 ;
04A8    CBDE        0677            SET   3,(HL)
04AA    DDCB05DE    0678            SET   3,(IX+IND)
04AE    FDCB05DE    0679            SET   3,(IY+IND)
04B2    CBDF        0680            SET   3,A
04B4    CBD8        0681            SET   3,B
04B6    CBD9        0682            SET   3,C
04B8    CBDA        0683            SET   3,D
04BA    CBDB        0684            SET   3,E
04BC    CBDC        0685            SET   3,H
04BE    CBDD        0686            SET   3,L
                    0687 ;
04C0    CBE6        0688            SET   4,(HL)
04C2    DDCB05E6    0689            SET   4,(IX+IND)
04C6    FDCB05E6    0690            SET   4,(IY+IND)
04CA    CBE7        0691            SET   4,A
04CC    CBE0        0692            SET   4,B
04CE    CBE1        0693            SET   4,C
04D0    CBE2        0694            SET   4,D
04D2    CBE3        0695            SET   4,E
04D4    CBE4        0696            SET   4,H
```

```
 LOC    OBJ CODE    STMT SOURCE STATEMENT

04D6    CBE5        0697            SET   4,L
                    0698 ;
04D8    CBEE        0699            SET   5,(HL)
04DA    DDCB05EE    0700            SET   5,(IX+IND)
04DE    FDCB05EE    0701            SET   5,(IY+IND)
04E2    CBEF        0702            SET   5,A
04E4    CBE8        0703            SET   5,B
04E6    CBE9        0704            SET   5,C
04E8    CBEA        0705            SET   5,D
04EA    CBEB        0706            SET   5,E
04EC    CBEC        0707            SET   5,H
04EE    CBED        0708            SET   5,L
                    0709 ;
04F0    CBF6        0710            SET   6,(HL)
04F2    DDCB05F6    0711            SET   6,(IX+IND)
04F5    FDCB05F6    0712            SET   6,(IY+IND)
04FA    CBF7        0713            SET   6,A
04FC    CBF0        0714            SET   6,B
04FE    CBF1        0715            SET   6,C
0500    CBF2        0716            SET   6,D
0502    CBF3        0717            SET   6,E
0504    CBF4        0718            SET   6,H
0506    CBF5        0719            SET   6,L
                    0720 ;
0508    CBFE        0721            SET   7,(HL)
050A    DDCB05FE    0722            SET   7,(IX+IND)
050E    FDCB05FE    0723            SET   7,(IY+IND)
0512    CBFF        0724            SET   7,A
0514    CBF8        0725            SET   7,B
0515    CBF9        0726            SET   7,C
0518    CBFA        0727            SET   7,D
051A    CBFB        0728            SET   7,E
051C    CBFC        0729            SET   7,H
051E    CBFD        0730            SET   7,L
                    0731 ;
0520    CB26        0732            SLA   (HL)
0522    DDCB0526    0733            SLA   (IX+IND)
0526    FDCB0526    0734            SLA   (IY+IND)
052A    CB27        0735            SLA   A
052C    CB20        0736            SLA   B
052E    CB21        0737            SLA   C
0530    CB22        0738            SLA   D
0532    CB23        0739            SLA   E
0534    CB24        0740            SLA   H
0536    CB25        0741            SLA   L
                    0742 ;
0538    CB2E        0743            SRA   (HL)
053A    DDCB052E    0744            SRA   (IX+IND)
053E    FDCB052E    0745            SRA   (IY+IND)
0542    CB2F        0746            SRA   A
0544    CB28        0747            SRA   B
0546    CB29        0748            SRA   C
0548    CB2A        0749            SRA   D
054A    CB2B        0750            SRA   E
054C    CB2C        0751            SRA   H
054E    CB2D        0752            SRA   L
                    0753 ;
0550    CB3E        0754            SRL   (HL)
```

```
 LOC   OBJ CODE   STMT SOURCE STATEMENT

0552   DDCB053E  0755           SRL   (IX+IND)
0556   FDCB053E  0756           SRL   (IY+IND)
055A   CB3F      0757           SRL   A
055C   CB38      0758           SRL   B
055E   CB39      0759           SRL   C
0560   CB3A      0760           SRL   D
0562   CB3B      0761           SRL   E
0564   CB3C      0762           SRL   H
0566   CB3D      0763           SRL   L
                 0764 ;
0568   96        0765           SUB   (HL)
0569   DD9605    0766           SUB   (IX+IND)
056C   FD9605    0767           SUB   (IY+IND)
056F   97        0768           SUB   A
0570   90        0769           SUB   B
0571   91        0770           SUB   C
0572   92        0771           SUB   D
0573   93        0772           SUB   E
0574   94        0773           SUB   H
0575   95        0774           SUB   L
0576   D620      0775           SUB   N
                 0776 ;
0578   AE        0777           XOR   (HL)
0579   DDAE05    0778           XOR   (IX+IND)
057C   FDAE05    0779           XOR   (IY+IND)
057F   AF        0780           XOR   A
0580   A8        0781           XOR   B
0581   A9        0782           XOR   C
0582   AA        0783           XOR   D
0583   AB        0784           XOR   E
0584   AC        0785           XOR   H
0585   AD        0786           XOR   L
0586   EE20      0787           XOR   N
                 0788 ;
0588             0789 NN:       DEFS  2
                 0790 IND:      EQU   5
                 0791 N:        EQU   20H
                 0792 DIS:      EQU   30H
                 0793 ;
058A   00        0794 R1:       DEFB  0
058B   0000      0795 R2:       DEFW  0
058D             0796 R3:       DEFS  1
                 0797 R4:       DEFL  R3
                 0798 R5:       EQU   R4
                 0799           END
```

APPENDIX B

MOSTEK ASSEMBLER STANDARD PSEUDO-OPS

APPENDIX B

MOSTEK ASSEMBLER STANDARD PSEUDO-OPS

B-1. INTRODUCTION.

B-2. The following pseudo-ops are standard for Z80 assemblers from MOSTEK. Note that other pseudo-ops may be allowed depending on the features of a particular assembler. For example, additional pseudo-ops may be required to handle conditional assembly, global symbols, and macros.

DEFB n

Define byte of memory.

Operation: (PC) ← n (static)

Format

Opcode	Operands	Machine Code
DEFB	n	← n →

(no execution time)

Description: This pseudo-op reserves and defines one byte of memory to contain the value n.

Example:

```
      DEFB    OAH
```

causes the current memory location to be defined with the value OAH.

label DEFL nn

Define 'label' to have the value nn.

Operation: label <- nn

Format:

	Opcode	Operand
label	DEFL	nn

(no execution time, no machine code)

Description: This pseudo-op assigns the value nn to the label which appears in the label field. The same label can be defined any number of times in a program using this pseudo-op.

Example:

```
LAB4:    DEFL    050AH
```

The label 'LAB4' is defined to have the value 050AH.

DEFM 's'

Define message

Operation: (PC) ← S_1

(PC+1) ← S_2

(PC+2) ← S_3

.

.

.

where s_1 is the first ASCII character in string s, s_2 is the second ASCII character, etc.

Format:

Opcode	Operand	Machine code
DEFM	's'	s_1
		s_2
		s_3
		.
		.
		.

(no execution time)

Description: This pseudo-op reserves and defines sequential bytes of memory to contain ASCII equivalents of the characters in the string s.

Example:

```
DEFM   'ABC'
```

will reserve 3 bytes of memory and cause them to be loaded with 41H,42H,43H, respectively.

DEFS nn

Define storage

Operation: (PC) ← (PC) + nn (static)

Format

Opcode	Operand	Machine code
DEFS	nn	

(no execution time)

Description: This pseudo-op causes nn bytes of memory to be defined as storage. In the object module, these bytes are not loaded. In a load (binary) module, these bytes are loaded with meaningless data.

Example:

```
DEFS   40D
```

This causes 40 (decimal) memory locations to be defined as storage and skipped in the object module.

DEFW nn

Define word of memory

Operation: (PC) ← nn + 1

(PC+1)← nn (static)

Format

Opcode	Operand	Machine code	
DEFW	nn	nn + 1	(Lower byte)
		nn	(Upper byte)

Description: This pseudo-op reserves and defines two bytes of memory. The first byte is defined to contain the least significant byte of the operand nn. The next byte is defined to contain the most significant byte of the operand nn.

Example:

```
DEFW    0A00H
```

will define the current memory location to contain 00H and the next memory location to contain 0AH.

END s

End of assembly

Operation: terminates current assembler pass.

Format

Opcode	Operand
END	s

(no execution time, no machine code)

Description: This pseudo-op terminates the current assember pass. The operand s is optional and is an expression which defines the starting execution address of the program being assembled. The value of s is entered in the end-of-file record in the object output of the assembler.

Example:

```
END   0AAH
```

terminates the current assembler pass and causes 0AAH to be defined as the starting address of the program.

label EQU nn

Equate 'label' to value nn.

Operation: label nn

Format:

	Opcode	Operand
label	EQU	nn

(no execution time, no machine code)

Description: This pseudo-op assigns the value nn to the label which appears in the label field. The label can only appear once in the label field in a program using this pseudo-op.

Example:

```
LAB4:    EQU  05H
```

The label 'LAB4' is defined to have the value 05H.

APPENDIX C

MOSTEK STANDARD Z80 OBJECT CODE FORMAT

APPENDIX C

MOSTEK STANDARD Z80 OBJECT OUTPUT DEFINITION

C-1. INTRODUCTION.

C-2. Each record of an object module begins with a delimiter (colon or dollar sign) and ends with carriage return and line feed. A colon (:) is used for data records and end of file record. A dollar sign ($) is used for records containing relocation information and linking information. An Intel loader will ignore such information and allow loading of non-relocatable, non-linkable programs. All information is in ASCII.

C-3. Each record is identified as a type . The type appears in the 8th and 9th bytes of the record and can take the following values:

00 - data record
01 - end-of-file
02 - internal symbol
03 - external symbol
04 - relocation information
05 - module definition

C-4. DATA RECORD FORMAT (TYPE 00).

Byte 1 Colon (:) delimiter

2-3 Number of binary bytes of data in this record. The maximum is 32 binary bytes (64 ASCII bytes).

4-5 Most significant byte of the start address of data.

6-7 Least significant byte of start address of data.

8-9 ASCII zeros. This is the "record type" for data.

10- Data bytes.

Last two bytes - Checksum of all bytes except the delimiter, carriage return, and line feed. The checksum is the negative of the binary sum of all bytes in the record.

CRLF Carriage return, line feed.

C-5. END-OF-FILE RECORD (TYPE 01).

Byte 1 Colon (:) delimiter.

2-3 ASCII zeros.

4-5 Most significant byte of the transfer address of the program. This transfer address appears as an argument in the 'END' pseudo-op of a program. It represents the starting execution address of the program.

6-7 Least significant byte of the transfer address.

8-9 Record type 01.

10-11 Checksum.

CRLF Carriage return, line feed.

C-6. INTERNAL SYMBOL RECORD (TYPE 02).

Byte 1 Dollar sign ($) delimiter.

2-7 Up to 6 ASCII characters of the internal symbol name. The name is left justified, blank filled.

8-9 Record type 02.

10-13 Address of the internal symbol, most significant byte first.

14-15 Binary checksum. Note that the ASCII letters of the symbol are converted to binary before th checksum is calculated. Binary conversion is done without regard to errors.

CRLF Carriage return, line feed.

C-7. EXTERNAL SYMBOL RECORD (TYPE 03).

Byte 1 Dollar sign ($) delimiter.

2-7 Up to 6 ASCII characters of the external symbol name. The name is left justified, blank filled.

8-9 Record type 03.

10-13 Last address which uses the external symbol. This is the start of a link list in the object data records which is described below. The most significant byte is first.

14-15 Binary checksum.

CRLF Carriage return, line feed.

C-8. The ASMB-80 Assembler outputs the external symbol name and the last address in the program where the symbol is used. The data records which follow contain a link list pointing to all occurrences of that symbol in the object code.

1. The external symbol record shows the symbol ('LAB') and the last location in the program which uses the sumbol (212AH).

2. The object code at 212AH has a pointer which shows where the previous reference to the external symbol occurred (200FH).

3. This backward reference list continues until a terminator ends the list. This terminator is OFFFFH.

This method is easy to generate and decode. It has the advantage of reducing the number of bytes of object code needed to define all external references in a program.

C-9. RELOCATING INFORMATION RECORD (TYPE 04)

Both the internally referenced relocatable addresses and the elements of the eternal global reference linked list will be defined in these records.

Byte 1	Dollar sign ($) delimiter.
2-3	Number of sets of 2 ASCII characters, where 2 sets define an address.
4-7	ASCII zeros.
8-9	Record type 04.
10-	Addresses which must be relocated, most significant byte first.

Last two bytes - Binary checksum.

CRLF	Carriage return, line feed.

C-10. MODULE DEFINITION RECORD (TYPE 05).

This record has the name of the module (defined by the 'NAME' pseudo-op) and a loading information flag byte. The flag byte is determined by the 'PSECT' pseudo-op.

Byte 1	Dollar sign ($) delimiter.
2-7	Name of the module, left justified, blank filled.
8-9	Record type 05.
10-11	Flag byte. When converted to binary, the flag byte is defined as follows:

Bit 0 - 0 for absolute assemblies
1 for relocatable assemblies

12-13	Binary checksum.
CRLF	Carriage return, line feed.

APPENDIX D

REFERENCE TABLES

TABLE D-1. Hexadecimal to Decimal Conversion Table

HEXADECIMAL COLUMNS											
6		5		4		3		2		1	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0	0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
0123		4567		0123		4567		0123		4567	
BYTE				BYTE				BYTE			

TABLE D-2. ASCII Character Set (7-Bit Code)

MSD / LSD		0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P		p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENG	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	
C	1100	FF	FS	,	<	L	\	l	{
D	1101	CR	GS	-	=	M	]	m	
E	1110	SO	RS	.	>	N	↑	n	~
F	1111	SI	VS	/	?	O	←	o	DEL

TABLE D-3. Powers of 2

2^n	n
256	8
512	9
1 024	10
2 048	11
4 096	12
8 192	13
16 384	14
32 768	15
65 536	16
131 072	17
262 144	18
524 288	19
1 048 576	20
2 097 152	21
4 194 304	22
8 388 608	23
16 777 216	24

TABLE D-4.

Powers Of 2/Powers Of 16 Conversion

$2^0 = 16^0$
$2^4 = 16^1$
$2^8 = 16^2$
$2^{12} = 16^3$
$2^{16} = 16^4$
$2^{20} = 16^5$
$2^{24} = 16^6$
$2^{28} = 16^7$
$2^{32} = 16^8$
$2^{36} = 16^9$
$2^{40} = 16^{10}$
$2^{44} = 16^{11}$
$2^{48} = 16^{12}$
$2^{52} = 16^{13}$
$2^{56} = 16^{14}$
$2^{60} = 16^{15}$

TABLE D-5. Powers of 16

16^n	n
1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7
4 294 967 296	8
68 719 476 736	9
1 099 511 627 776	10
17 592 186 044 416	11
281 474 976 710 656	12
4 503 599 627 370 496	13
72 057 594 037 927 936	14
1 152 921 504 606 846 976	15

MOSTEK 280 PROGRAMMING MANUAL

MOSTEK®
Z80·F8 Covering the full spectrum of
3870 microcomputer applications.

1215 W. Crosby Rd. • Carrollton, Texas 75006 • 214/242-0444
In Europe, contact: MOSTEK GmbH, Talstrasse 172
D 7024 Filderstadt-1, W. Germany • Tele: (0711) 701096

Reprinted in England by Valentine Press Ltd., 56 Woodford Avenue, Ilford, Essex, England August 1978

Publication No. MK 78515