Z80 Assembly Language

Notes about assembly language on the Z80

Peter Mount, Area51.dev & Contributors

Z80 Assembly Language

Notes about assembly language on the Z80

Title	Z80 Assembly Language
Subtitle	Notes about assembly language on the Z80
Author	Peter Mount, Area51.dev & Contributors
Copyright	CC BY-SA

Table of Contents

This section covers assembly language for the Z80 Microprocessor used on machines like the ZX Spectrum, Amstrad CPC and CP/M based machines.

1 - About the Z80

About the Z80

The Z80 is an 8-bit microprocessor introduced by Zilog as the startup company's first product. The Z80 was conceived by Federico Faggin in late 1974 and developed by him and his 11 employees starting in early 1975. The first working samples were delivered in March 1976, and it was officially introduced on the market in July 1976. With the revenue from the Z80, the company built its own chip factories and grew to over a thousand employees over the following two years.

The Zilog Z80 is a software-compatible extension and enhancement of the Intel 8080 and, like it, was mainly aimed at embedded systems. Although used in that role, the Z80 also became one of the most widely used CPUs in desktop computers and home computers from the 1970s to the mid-1980s. It was also common in military applications, musical equipment such as synthesizers (like the Roland Jupiter-8), and coin operated arcade games of the late 1970s and early 1980s including Pac-Man.

1.1 - Z80 Registers

About the Registers available on the Z80

The Z80 contains 208 bits of memory that are available to the programmer as registers.

Z80 Registers

Register Set				Special Purpose Registers	
Main		Alternate			
Accumulator A	Flags F	Accumulator A'	Flags F'	Interrupt Vector I	Memory Refresh R
B	C	B'	C'	Index Register IX	Index Register IY
D	E	D'	E'	Stack Pointer SP	
H	H	H'	L'	Program Counter PC	

Accumulator and Flag registers

The Z80 provides two independent 8-bit accumulators each with an associated flag register. The programmer can switch between the two pairs with the EX AF, AF' instruction.

General Purpose registers

Two matched sets of general purpose registers are available, each set containing six 8-bit registers: B, C, D, E, H and L.

These registers are also arranged to provide 3 16-bit registers: BC, DE and HL.

The HL register pair is usually used for addressing memory and has more instructions available to it for this purpose than BC or DE register pairs.

The programmer can switch between the main (BC, DE and HL) and alternate (BC', DE' and HL') set of general purpose registers with the EXX instruction.

PC Program Counter

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

SP Stack Pointer

The stack pointer holds the 16-bit address of the current top of a stack located anywhere in external system RAM. The external stack memory is organized as a last-in first-out (LIFO) file. Data can be pushed onto the stack using the PUSH instructions or popped off of the stack using the POP instructions.

1.2 - Z80 Status Flags

The Flag registers, F and F', supply information to the user about the status of the Z80 CPU at any particular time. Each of these two Flag registers contains 6 bits of status information that are set or cleared by CPU operations; bits 3 and 5 are not used.

Four of these bits (C, P/V, Z, and S) can be tested for use with conditional JUMP, CALL, or RETURN instructions.

The H and N flags cannot be tested; these two flags are used for BCD arithmetic.

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

C Carry

The Carry Flag (C) is set or cleared depending on the operation being performed.

For ADD instructions that generate a Carry, and for SUB instructions that generate a Borrow, the Carry Flag is set.

The Carry Flag is reset by an ADD instruction that does not generate a Carry, and by a SUB instruction that does not generate a Borrow.

This saved Carry facilitates software routines for extended precision arithmetic.

Additionally, the DAA instruction sets the Carry Flag if the conditions for making the decimal adjustment are met.

For the RLA, RRA, RLS, and RRS instructions, the Carry bit is used as a link between the least-significant byte (LSB) and the most-significant byte (MSB) for any register or memory location. During the RLCA, RLC, and SLA instructions, the Carry flag contains the final value shifted out of bit 7 of any register or memory location. During the RRCA, RRC, SRA, and SRL instructions, the Carry flag contains the final value shifted out of bit 0 of any register or memory location.

For the logical instructions AND, OR, and XOR, the Carry flag is reset.

The Carry flag can also be set by the Set Carry Flag (SCF) instruction and complemented by the Compliment Carry Flag (CCF) instruction.

Z Zero

The Zero Flag (Z) is set (1) or cleared (0) if the result generated by the execution of certain instructions is 0.

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

For Compare (search) instructions, the Z flag is set to 1 if the value in the Accumulator is equal to the value in the memory location indicated by the value of the register pair HL.

When testing a bit in a register or memory location, the Z flag contains the complemented state of the indicated bit.

When inputting or outputting a byte between a memory location and an INI, IND, OUTI, or OUTD I/O device, if the result of decrementing Register B is 0, then the Z flag is 1; otherwise, the Z flag is 0. Additionally, for byte inputs from I/O devices using IN r, (C), the Z flag is set to indicate a 0-byte input.

P/V Parity Overflow

The Parity/Overflow (P/V) Flag is set to a specific state depending on the operation being performed.

Overflow

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 is determined by examining the sign bits of the operands.

For addition, operands with different signs never cause overflow. When adding operands with similar signs and the result contains a different sign, the Overflow Flag is set.

For subtraction, overflow can occur for operands of unalike signs. Operands of alike signs never cause overflow.

Another method for identifying an overflow is to observe the Carry to 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 an Overflow has occurred.

Parity

This flag is also used with logical operations and rotate instructions to indicate the resulting parity is even. The number of 1 bits in a byte are counted. If the total is Odd, ODD parity is flagged (i.e., P = 0). If the total is even, even parity is flagged (i.e., P = 1).

When inputting a byte from an I/O device with an IN r, (C) instruction, the P/V Flag is adjusted to indicate data parity.

Alternate usage

During the CPI, CPIR, CPD, and CPDR search instructions and the LDI, LDIR, LDD, and LDDR block transfer instructions, the P/V Flag monitors the state of the Byte Count (BC) Register. When decrementing, if the byte counter decrements to 0, the flag is cleared to 0; otherwise the flag is set to 1.

During the LD A, I and LD A, R instructions, the P/V Flag is set with the value of the interrupt enable flip-flop (IFF2) for storage or testing.

S Sign

The Sign Flag (S) stores the state of the most-significant bit of the Accumulator (bit 7). When the Z80 CPU performs arithmetic operations on signed numbers, the binary twos-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 twos complement of the equivalent positive number. The total range for negative numbers is from –1 to –128.

When inputting a byte from an I/O device to a register using an IN r, (C) instruction, the S Flag indicates either positive (S = 0) or negative (S = 1) data.

N Add/Subtract

The Add/Subtract Flag (N) is used by the Decimal Adjust Accumulator instruction (DAA) to distinguish between the ADD and SUB instructions.

For ADD instructions, N is cleared to 0. For SUB instructions, N is set to 1.

H Half Carry

The Half Carry Flag (H) is set (1) or cleared (0) 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 (DAA) instruction to correct the result of a packed BCD add or subtract operation.

For ADD instructions, H is set if a carry occurs from bit 3 to bit 4. For SUB instructions, H is set if a borrow from bit 4 occurs.

1.3 - Addresses

The addresses used ny the Z80

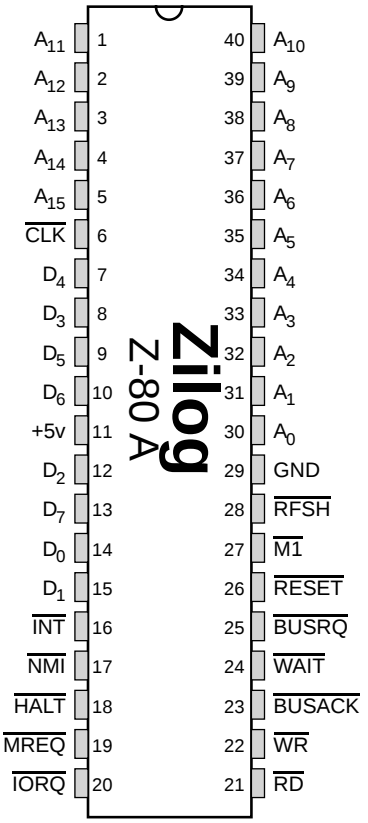
The Z80 uses a fixed set of addresses in Page 0 of the address space:

Address	Instruction	Usage
0000	RST 0	Initial power on `RST 0` instruction is invoked. $\overline{\text{RESET}}$ pin is held low
0008	RST 1	`RST 1` instruction is invoked.
0010	RST 2	`RST 2` instruction is invoked.
0018	RST 3	`RST 3` instruction is invoked.
0020	RST 4	`RST 4` instruction is invoked.
0028	RST 5	`RST 5` instruction is invoked.
0030	RST 6	`RST 6` instruction is invoked.
0038	RST 7	`RST 7` instruction is invoked. $\overline{\text{INT}}$ Maskable Interrupt handler when in Interrupt Mode 1
0066		$\overline{\text{NMI}}$ interrupt handler

Addresses 0x0000...0x003F are used by the 8 `RST` instructions with 8 bytes available for each. `RST 0` is also the start address for when the processor powers on or is reset.

1.4 - Pin Layout

The physical Z80 processor



General pins

A_0...A_{15} Address Bus

Address Bus (output, active High, tristate). A0...A15 form a 16-bit Address Bus, which provides the addresses for memory data bus exchanges (up to 64 KB) and for I/O device exchanges.

D_0...D_7 Data Bus

D_0...D_7 constitute an 8-bit bidirectional data bus, used for data exchanges with memory and I/O.

$\overline{\text{CLK}}$ Clock (input)

Single-phase MOS-level clock.

System Control

$\overline{\text{M1}}$ Machine Cycle One (output, active Low)

$\overline{\text{M1}}$, together with $\overline{\text{MREQ}}$, indicates that the current machine cycle is the op code fetch cycle of an instruction execution. $\overline{\text{M1}}$, when operating together with $\overline{\text{IORQ}}$, indicates an interrupt acknowledge cycle.

$\overline{\text{MREQ}}$ Memory Request (output, active Low, tristate)

$\overline{\text{MREQ}}$ indicates that the address bus holds a valid address for a memory read or a memory write operation.

$\overline{\text{IORQ}}$ Input/Output Request (output, active Low, tristate)

$\overline{\text{IORQ}}$ indicates that the lower half of the address bus holds a valid I/O address for an I/O read or write operation. $\overline{\text{IORQ}}$ is also generated concurrently with $\overline{\text{M1}}$ during an interrupt acknowledge cycle to indicate that an interrupt response vector can be placed on the data bus.

$\overline{\text{RD}}$ Read (output, active Low, tristate)

$\overline{\text{RD}}$ indicates that the CPU wants to read data from memory or an I/O device. The addressed I/O device or memory should use this signal to gate data onto the CPU data bus.

$\overline{\text{WR}}$ Write (output, active Low, tristate)

$\overline{\text{WR}}$ indicates that the CPU data bus contains valid data to be stored at the addressed memory or I/O location.

$\overline{\text{RFSH}}$ Refresh (output, active Low)

$\overline{\text{RFSH}}$, together with $\overline{\text{MREQ}}$, indicates that the lower seven bits of the system's address bus can be used as a refresh address to the system's dynamic memories.

Bus Control

$\overline{\text{BUSACK}}$ Bus Acknowledge

The $\overline{\text{BUSACK}}$ pin indicates to the requesting device that the CPU address bus, data bus and control signals $\overline{\text{MREQ}}$, $\overline{\text{IORQ}}$, $\overline{\text{RD}}$ and $\overline{\text{WR}}$ have entered their high-impedance states and other devices on the bus can control those lines.

$\overline{\text{BUSREQ}}$ Bus Request

$\overline{\text{BUSREQ}}$ contains a higher priority than NMI and is always recognized at the end of the current machine cycle.

$\overline{\text{BUSREQ}}$ forces the CPU address bus, data bus, and control signals $\overline{\text{MREQ}}$, $\overline{\text{IORQ}}$, $\overline{\text{RD}}$ and $\overline{\text{WR}}$ to enter a high-impedance state so that other devices can control these lines. $\overline{\text{BUSREQ}}$ is normally wired OR and requires an external pull-up for these applications.

Extended $\overline{\text{BUSREQ}}$ periods due to extensive DMA operations can prevent the CPU from properly refreshing dynamic RAM.

CPU Control

HALT HALT State (output, active Low)

HALT indicates that the CPU has executed a HALT instruction and is waiting for either a nonmaskable or a maskable interrupt (with the mask enabled) before operation can resume. During HALT, the CPU executes NOPs to maintain memory refreshes.

WAIT WAIT (input, active Low)

WAIT communicates to the CPU that the addressed memory or I/O devices are not ready for a data transfer. The CPU continues to enter a WAIT state as long as this signal is active. Extended WAIT periods can prevent the CPU from properly refreshing dynamic memory.

INT Interrupt Request (input, active Low)

An Interrupt Request is generated by I/O devices. The CPU honors a request at the end of the current instruction if the internal software-controlled interrupt enable flip-flop (IFF) is enabled. INT is normally wired-OR and requires an external pull-up for these applications.

NMI Nonmaskable Interrupt (input, negative edge-triggered)

NMI contains a higher priority than INT. NMI is always recognized at the end of the current instruction, independent of the status of the interrupt enable flip-flop, and automatically forces the CPU to restart at location 0066h.

RESET Reset (input, active Low)

RESET initializes the CPU as follows:

- it resets the interrupt enable flip-flop,
- clears the Program Counter and registers I and R,
- sets the interrupt status to Mode 0.

During reset time, the address and data bus enter a high-impedance state, and all control output signals enter an inactive state. RESET must be active for a minimum of three full clock cycles before a reset operation is complete.

2 - Timing

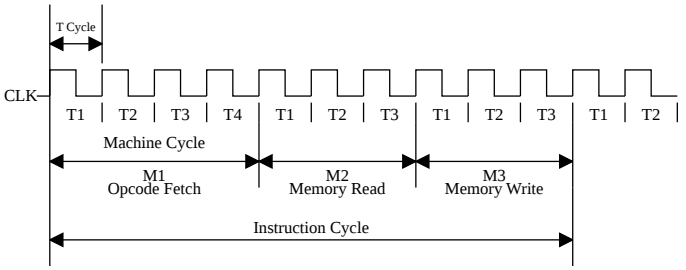
How the system clock relates to processor speed

The Z80 processor executes instructions as a series of basic operations:

- Memory access
- I/O device access
- Interrupt acknowledge

Each of these operations is known as a Machine cycle (`M-cycle`), which can take between 3 and 6 `T-Cycles` to execute, although this can be extended by the $\overline{\text{WAIT}}$ signal. A `T-cycle` (Time Cycle) is one cycle of the system clock.

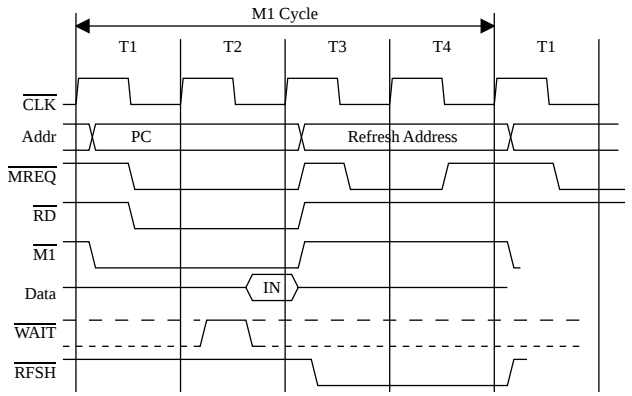
The following diagram shows an example of a single instruction that reads from memory and writes back.



Basic CPU Timing Example

Opcode fetch

The Opcode fetch takes 4 T-cycles:



Instruction Opcode Fetch

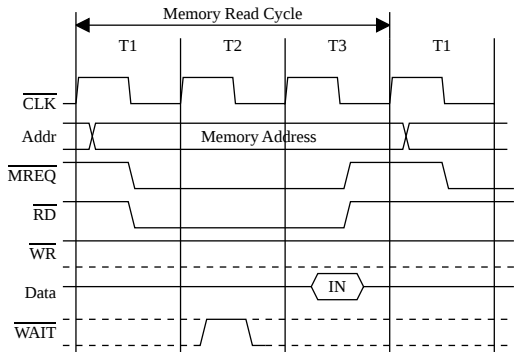
T1 Sets the Address bus $A_{0...15}$ to the current value of the program counter.
T2 Reads the opcode during the second half of the cycle.
T3 and T4 has the refresh address set on the Address Bus. This is to allow dynamic ram to be refreshed.

Memory Access

Memory access cycles are generally three T-cycles long unless wait states are requested by memory via the $\overline{\text{WAIT}}$ signal.

Memory Read Cycle

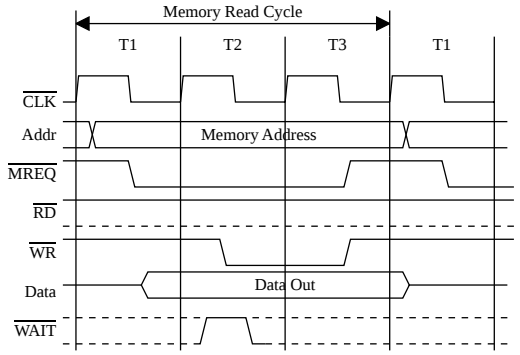
For a memory read the $\overline{\text{MREQ}}$ and $\overline{\text{RD}}$ signals are pulled low once the address bus is stable.



Memory Read Cycle

Memory Write Cycle

For a memory write the $\overline{\text{MREQ}}$ and $\overline{\text{WR}}$ signals are pulled low once the address bus is stable.



Memory Write Cycle

$\overline{\text{WR}}$ goes inactive half a T-State before the address and data bus contents are changed to support different types of memory.

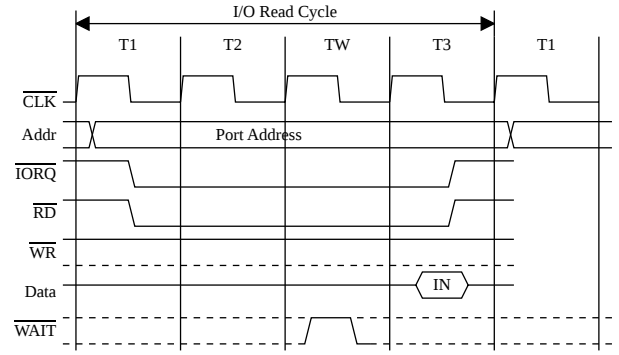
I/O Cycles

I/O operations are similar to memory, except the $\overline{\text{IORQ}}$ signal is used instead of $\overline{\text{MREQ}}$ to indicate that devices not memory should respond.

Also an additional wait state is inserted after T-State 2. The reason for this single wait state insertion is that during I/O operations, the period from when the $\overline{\text{IORQ}}$ signal goes active until the CPU must sample the $\overline{\text{WAIT}}$ line is short. Without this extra state, sufficient time does not exist for an I/O port to decode its address and activate the $\overline{\text{WAIT}}$ line if a wait is required. Additionally, without this wait state, it is difficult to design MOS I/O devices that can operate at full CPU speed. During this wait state period, the $\overline{\text{WAIT}}$ request signal is sampled.

I/O Read Cycle

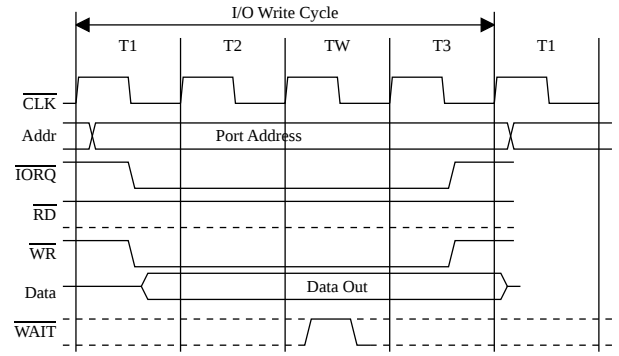
During a read I/O operation, the $\overline{\text{RD}}$ line is used to enable the addressed port onto the data bus, just as in the case of a memory read.



I/O Read Cycle

I/O Write Cycle

During a write I/O operation, the $\overline{\text{WR}}$ line is used to enable the addressed port onto the data bus, just as in the case of a memory write.



I/O Write Cycle

3 - Opcodes

Instruction Set

Instruction Notation Summary

Notation	Description
r	Identifies any of the registers **A**, **B**, **C**, **D**, **E**, **H** or **L**
(*HL*)	Identifies the contents of the memory location whose address is specified by the contents of the **HL** register pair.
(*IX* + *d*)	Identifies the contents of the memory location, whose address is specified by the contents of the Index register pair **IX** plus the signed displacement **d**
(*IY* + *d*)	Identifies the contents of the memory location, whose address is specified by the contents of the Index register pair **IY** plus the signed displacement **d**
n	Identifies a one-byte unsigned integer expression in the range (0 to 255)
nn	Identifies a two-byte unsigned integer expression in the range (0 to 65535) (0x0000 to 0xFFFF)
b	Identifies a one-byte signed integer expression in the range (-128 to +127)
e	Identifies a one-byte signed integer expression in the range (-126 to +129) for relative jump offset from current location
cc	Identifies the status of the Flag Register as any of (**NZ**, **Z**, **NC**, **C**, **PO**, **PE**, **P** or **M**) for the conditional jumps, calls, and return instructions
qq	Identifies any of the register pairs **BC**, **DE**, **HL** or **AF**
ss	Identifies any of the register pairs **BC**, **DE**, **HL** or **SP**
pp	Identifies any of the register pairs **BC**, **DE**, **IX** or **SP**
rr	Identifies any of the register pairs **BC**, **DE**, **IY** or **SP**
s	Identifies any of **r**, **n**, **(HL)**, **(IX+d)** or **(IY+d)**
m	Identifies any of **r**, **(HL)**, **(IX+d)** or **(IY+d)**

3.1 - Load

Load registers, data & memory

3.1.1 - LD (dd), n

Load number into memory

7	6	5	4	3	2	1	0

$(HL) \longleftarrow n$

LD (HL), n

0	0	1	1	0	1	1	0	36
n								

$(IX + d) \longleftarrow n$

LD (IX+d), n

1	1	0	1	1	1	0	1	DD
0	0	1	1	0	1	1	0	36
d								
n								

$(IY + d) \longleftarrow n$

LD (IY+d), n

1	1	1	1	1	1	0	1	FD
0	0	1	1	0	1	1	0	36
d								
n								

Flags Affected

None.

Opcode Matrix

	(HL)	(IX+d)	(IY+d)
n	LD (HL), n 2 10 36nn	LD (IX+d), n 4 19 DD36nnnn	LD (IY+d), n 4 19 FD36nnnn

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Implicit

3.1.2 - LD (dd), s

Store register into memory via register

7	6	5	4	3	2	1	0

$(BC) \longleftarrow A$

LD (BC), A

0	0	0	0	0	0	1	0	02

$(DE) \longleftarrow A$

LD (DE), A

0	0	0	1	0	0	1	0	12

$(HL) \longleftarrow r$

LD (HL), r

0	1	1	1	0	r	

$(IX + d) \longleftarrow r$

LD (IX+d), r

1	1	0	1	1	1	0	1	DD
0	1	1	1	0	r			
d								

$(IY + d) \longleftarrow r$

LD (IY+d), r

1	1	1	1	1	1	0	1	FD
0	1	1	1	0	r			
d								

Registers

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

Flags Affected

None.

Opcode Matrix

	A	B	C	D	E	H	L
(HL)	LD (HL), A 1 7 77	LD (HL), B 1 7 70	LD (HL), C 1 7 71	LD (HL), D 1 7 72	LD (HL), E 1 7 73	LD (HL), H 1 7 74	LD (HL), L 1 7 75
(BC)	LD (BC), A 1 7 02						
(DE)	LD (DE), A 1 7 12						
(IX+d)	LD (IX+d), A 3 19 DD77nn	LD (IX+d), B 3 19 DD70nn	LD (IX+d), C 3 19 DD71nn	LD (IX+d), D 3 19 DD72nn	LD (IX+d), E 3 19 DD73nn	LD (IX+d), H 3 19 DD74nn	LD (IX+d), L 3 19 DD75nn
(IY+d)	LD (IY+d), A 3 19 FD77nn	LD (IY+d), B 3 19 FD70nn	LD (IY+d), C 3 19 FD71nn	LD (IY+d), D 3 19 FD72nn	LD (IY+d), E 3 19 FD73nn	LD (IY+d), H 3 19 FD74nn	LD (IY+d), L 3 19 FD75nn

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Memory

3.1.3 - LD (nn), s

Store register into memory via address

7	6	5	4	3	2	1	0

$(nn) \longleftarrow A$

LD (nn), A

0	0	1	1	0	0	1	0	32
7 15				nn			0 8	

$(nn + 1) \longleftarrow dd_h, (nn) \longleftarrow dd_l$

LD (nn), dd

1	1	1	0	1	1	0	1	ED
0	1	dd		0	0	1	1	
7 15				nn			0 8	

$(nn + 1) \longleftarrow H, (nn) \longleftarrow L$

LD (nn), HL

0	0	1	0	0	0	1	0	22
7 15				nn			0 8	

$(nn + 1) \longleftarrow IX_h, (nn) \longleftarrow IX_l$

LD (nn), IX

1	1	0	1	1	1	0	1	DD
0	0	1	0	0	0	1	0	22
7 15				nn			0 8	

$(nn + 1) \longleftarrow IY_h, (nn) \longleftarrow IY_l$

LD (nn), IY

1	1	1	1	1	1	0	1	FD
0	0	1	0	0	0	1	0	22
7 15				nn			0 8	

Registers

Value	dd
00	BC
01	DE
10	HL
11	SP

Flags Affected

None.

Opcode Matrix

	A	BC	DE	HL	IX	IY	SP
(nn)	LD (nn), A 3 13 32nnnn			LD (nn), HL 3 16 22nnnn			
(nn)		LD (nn), BC 4 20 ED43nnnn	LD (nn), DE 4 20 ED53nnnn	LD (nn), HL 4 20 ED63nnnn	LD (nn), IX 4 20 DD22nnnn	LD (nn), IY 4 20 FD22nnnn	LD (nn), SP 4 20 ED73nnnn

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Memory

3.1.4 - LD A,I and LDA A,R

8-bit register instructions

7	6	5	4	3	2	1	0

$A \longleftarrow I$

LD A, I

1	1	1	0	1	1	0	1	ED
0	1	0	1	0	1	1	1	57

$A \longleftarrow R$

LD A, R

1	1	1	0	1	1	0	1	ED
0	1	0	1	1	1	1	1	5F

Flags Affected

Flags	s	z	-	-	-	p/v	-	-

s Set if the source register is negative

z Set if the source register is 0

p/v Contains contents of IFF2,
0 if an interrupt occurs during the instruction running

Opcode Matrix

	I	R
A	LD A, I 2 9 ED57	LD A, R 2 9 ED5F

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Special

3.1.5 - LD dd, nn

Load 16-bit number

7	6	5	4	3	2	1	0

$dd \longleftarrow nn$

LD dd, nn

0	0	dd	0	0	0	1	
7 15	nn						0 8

$IX \longleftarrow nn$

LD IX, nn

1	1	0	1	1	1	0	1	DD
0	0	1	0	0	0	0	1	21
7 15	nn						0 8	

$IY \longleftarrow nn$

LD IY, nn

1	1	1	1	1	1	0	1	FD
0	0	1	0	0	0	0	1	21
7 15	nn						0 8	

Registers

Value	dd
00	BC
01	DE
10	HL
11	SP

Flags Affected

None.

Opcode Matrix

	BC	DE	HL	IX	IY	SP
nn	LD BC, nn 3 10 01nnnn	LD DE, nn 3 10 11nnnn	LD HL, nn 3 10 21nnnn	LD IX, nn 4 14 DD21nnnn	LD IY, nn 4 14 FD21nnnn	LD SP, nn 3 10 31nnnn

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Implicit

3.1.6 - LD r, s

8-bit register instructions

7	6	5	4	3	2	1	0

$r \longleftarrow r'$

LD r, r'

0	1	r	r'

$r \longleftarrow n$

LD r, n

0	0	r	1	1	0
n					

$A \longleftarrow (BC)$

LD A, (BC)

0	0	0	0	1	0	1	0	0A

$A \longleftarrow (DE)$

LD A, (DE)

0	0	0	1	1	0	1	0	1A

$r \longleftarrow (HL)$

LD r, (HL)

0	1	r	1	1	0

$r \longleftarrow (IX + d)$

LD r, (IX+d)

1	1	0	1	1	1	0	1	DD
0	1	r		1	1	0		
d								

$r \longleftarrow (IY + d)$

LD r, (IY+d)

1	1	1	1	1	1	0	1	FD
0	1	r		1	1	0		
d								

$I \longleftarrow A$

LD I,A

1	1	1	0	1	1	0	1	ED
0	1	0	0	0	1	1	1	47

$R \longleftarrow A$

LD R, A

1	1	1	0	1	1	0	1	ED
0	1	0	0	1	1	1	1	4F

Registers

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

Flags Affected

None.

Opcode Matrix

	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX+d)	(IY+d)	n
A	LD A, A 1 4 7F	LD A, B 1 4 78	LD A, C 1 4 79	LD A, D 1 4 7A	LD A, E 1 4 7B	LD A, H 1 4 7C	LD A, L 1 4 7D	LD A, (HL) 1 7 7E	LD A, (BC) 1 7 0A	LD A, (DE) 1 7 1A	LD A, (IX+d) 3 19 DD7Enn	LD A, (IY+d) 3 19 FD7Enn	LD A, n 2 7 3Enn
B	LD B, A 1 4 47	LD B, B 1 4 40	LD B, C 1 4 41	LD B, D 1 4 42	LD B, E 1 4 43	LD B, H 1 4 44	LD B, L 1 4 45	LD B, (HL) 1 7 46			LD B, (IX+d) 3 19 DD46nn	LD B, (IY+d) 3 19 FD46nn	LD B, n 2 7 06nn
C	LD C, A 1 4 4F	LD C, B 1 4 48	LD C, C 1 4 49	LD C, D 1 4 4A	LD C, E 1 4 4B	LD C, H 1 4 4C	LD C, L 1 4 4D	LD C, (HL) 1 7 4E			LD C, (IX+d) 3 19 DD4Enn	LD C, (IY+d) 3 19 FD4Enn	LD C, n 2 7 0Enn
D	LD D, A 1 4 57	LD D, B 1 4 50	LD D, C 1 4 51	LD D, D 1 4 52	LD D, E 1 4 53	LD D, H 1 4 54	LD D, L 1 4 55	LD D, (HL) 1 7 56			LD D, (IX+d) 3 19 DD56nn	LD D, (IY+d) 3 19 FD56nn	LD D, n 2 7 16nn
E	LD E, A 1 4 5F	LD E, B 1 4 58	LD E, C 1 4 59	LD E, D 1 4 5A	LD E, E 1 4 5B	LD E, H 1 4 5C	LD E, L 1 4 5D	LD E, (HL) 1 7 5E			LD E, (IX+d) 3 19 DD5Enn	LD E, (IY+d) 3 19 FD5Enn	LD E, n 2 7 1Enn
H	LD H, A 1 4 67	LD H, B 1 4 60	LD H, C 1 4 61	LD H, D 1 4 62	LD H, E 1 4 63	LD H, H 1 4 64	LD H, L 1 4 65	LD H, (HL) 1 7 66			LD H, (IX+d) 3 19 DD66nn	LD H, (IY+d) 3 19 FD66nn	LD H, n 2 7 26nn
L	LD L, A 1 4 6F	LD L, B 1 4 68	LD L, C 1 4 69	LD L, D 1 4 6A	LD L, E 1 4 6B	LD L, H 1 4 6C	LD L, L 1 4 6D	LD L, (HL) 1 7 6E			LD L, (IX+d) 3 19 DD6Enn	LD L, (IY+d) 3 19 FD6Enn	LD L, n 2 7 2Enn
I	LD I, A 2 4 ED47												
R	LD R, A 2 4 ED4F												

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register	Memory	Implicit	Special

3.1.7 - LD s, (nn)

Load register from memory

7	6	5	4	3	2	1	0

$A \longleftarrow (nn)$

LD A, (nn)

0	0	1	1	1	0	1	0	3A
7 15				nn			0 8	

$H \longleftarrow (nn+1), L \longleftarrow (nn)$

LD HL, (nn)

0	0	1	0	1	0	1	0	2A
7 15				nn			0 8	

$dd_h \longleftarrow (nn+1), dd_l \longleftarrow (nn)$

LD dd, (nn)

1	1	1	0	1	1	0	1	ED
0	1	dd		1	0	1	1	
7 15				nn			0 8	

$IX_h \longleftarrow (nn+1), IX_l \longleftarrow (nn)$

LD IX, (nn)

1	1	0	1	1	1	0	1	DD
0	0	1	0	1	0	1	0	2A
7 15				nn			0 8	

$IY_h \longleftarrow (nn+1), IY_l \longleftarrow (nn)$

LD IY, (nn)

1	1	1	1	1	1	0	1	FD
0	0	1	0	1	0	1	0	2A
7 15				nn			0 8	

Registers

Value	dd
00	BC
01	DE
10	HL
11	SP

Flags Affected

None.

Opcode Matrix

	A	BC	DE	HL	IX	IY	SP
(nn)	LD A, (nn) 3 13 3Annnn			LD HL, (nn) 3 16 2Annnn			
(nn)		LD BC, (nn) 4 20 ED4Bnnnn	LD DE, (nn) 4 20 ED5Bnnnn	LD HL, (nn) 4 20 ED6Bnnnn	LD IX, (nn) 4 20 DD2Annnn	LD IY, (nn) 4 20 FD2Annnn	LD SP, (nn) 4 20 ED7Bnnnn

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Memory

3.1.8 - LD SP, s

Set Stack Pointer from register

7	6	5	4	3	2	1	0

$SP \longleftarrow HL$

LD SP,HL

1	1	1	1	1	0	0	1	F9

$SP \longleftarrow IX$

LD SP, IX

1	1	0	1	1	1	0	1	DD
1	1	1	1	1	0	0	1	F9

$SP \longleftarrow IY$

LD SP, IY

1	1	1	1	1	1	0	1	FD
1	1	1	1	1	0	0	1	F9

Flags Affected

None.

Opcode Matrix

	HL	IX	IY
SP	LD SP, HL 1 6 F9	LD SP, IX 2 6 DDF9	LD SP, IY 2 6 FDF9

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register

3.2 - Arithmetic

Arithmetic

3.2.1 - ADD without carry

Addition without carry

The ADD instruction performs an addition without carry. Any overflow from the addition will be passed on to the carry flag.

3.2.1.1 - ADD r without carry

Addition of a register without carry

7	6	5	4	3	2	1	0

$A \longleftarrow A + r$

ADD A, r

1	0	0	0	0	r

Registers

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

Flags Affected

Flags	s	z	-	h	-	-	-	-

s set if result negative

z set if result is 0

h set if carry from bit 3

Opcode Matrix

	A	B	C	D	E	H	L
A	ADD A,A 1 4 87	ADD A,B 1 4 80	ADD A,C 1 4 81	ADD A,D 1 4 82	ADD A,E 1 4 83	ADD A,H 1 4 84	ADD A,L 1 4 85

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register

3.2.1.2 - ADD n without carry

Addition of a number without carry

7	6	5	4	3	2	1	0

$A \longleftarrow A + n$

ADD A, n

1	1	0	0	0	1	1	0	C6
n								

Flags Affected

Flags	s	z	-	h	-	p/v	-	c

s set if result negative

z set if result is 0

h set if carry from bit 3

p/v set if overflow

c set if carry from bit 7

Opcode Matrix

	n
A	ADD A,n 2 7 C6nn

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Implicit

3.2.1.3 - ADD (dd) without carry

Addition of memory without carry

7	6	5	4	3	2	1	0

$A \longleftarrow A + (HL)$

ADD A, (HL)

1	0	0	0	0	1	1	0	86

$A \longleftarrow A + (IX + d)$

ADD A, (IX+d)

1	1	0	1	1	1	0	1	DD
1	0	0	0	0	1	1	0	86
d								

$A \longleftarrow A + (IY + d)$

ADD A, (IY+d)

1	1	1	1	1	1	0	1	FD
1	0	0	0	0	1	1	0	86
d								

Flags Affected

Flags	s	z	-	h	-	p/v	-	c

s set if result negative

z set if result is 0

h set if carry from bit 3

p/v set if overflow

c set if carry from bit 7

Opcode Matrix

	(HL)	(IX+d)	(IY+d)
A	ADD A,(HL) 1 7 86	ADD A,(IX+d) 3 19 DD86nn	ADD A,(IY+d) 3 19 FD86nn

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Memory

3.2.1.4 - ADD ss to HL without carry

Addition without carry

7	6	5	4	3	2	1	0

$HL \longleftarrow HL + dd$

ADD HL, dd

0	0	dd		1	0	0	1

$IX \longleftarrow IX + pp$

ADD IX, pp

1	1	0	1	1	1	0	1	DD
0	0	pp		1	0	0	1	

$IY \longleftarrow IY + mm$

ADD IY, mm

1	1	1	1	1	1	0	1	FD
0	0	mm		1	0	0	1	

Registers

Value	dd	mm	pp
00	BC	BC	BC
01	DE	DE	DE
10	HL	IY	IX
11	SP	SP	SP

Flags Affected

Flags	s	z	-	h	-	-	-	c

- **s** set if result negative
- **z** set if result is 0
- **h** set if carry from bit 11
- **c** set if carry from bit 15

Opcode Matrix

	BC	DE	HL	SP	IX	IY
HL	ADD HL,BC 1 11 09	ADD HL,DE 1 11 19	ADD HL,HL 1 11 29	ADD HL,SP 1 11 39		
IX	ADD IX,BC 2 15 DD09	ADD IX,DE 2 15 DD19		ADD IX,SP 2 15 DD39	ADD IX,IX 2 15 DD29	
IY	ADD IY,BC 2 15 FD09	ADD IY,DE 2 15 FD19		ADD IY,SP 2 15 FD39		ADD IY,IY 2 15 FD29

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register

3.2.2 - ADC Add with Carry

Addition with carry

The ADC instruction performs an addition with carry. If carry is set then it will be included in the calculation whilst any overflow from the addition will be passed on to the carry flag.

3.2.2.1 - ADC 8 bit add with Carry

Addition with carry

7	6	5	4	3	2	1	0

$A \longleftarrow A + r + Carry$

ADC A,r

1	0	0	0	1	r

$A \longleftarrow A + n + Carry$

ADC A,n

1	1	0	0	1	1	1	0	CE
n								

$A \longleftarrow A + (HL) + Carry$

ADC A, (HL)

1	0	0	0	1	1	1	0	8E

$A \longleftarrow A + (IX + d) + Carry$

ADC A, (IX + d)

1	1	0	1	1	1	0	1	DD
1	0	0	0	1	1	1	0	8E
d								

$A \longleftarrow A + (IY + d) + Carry$

ADC A, (IY + d)

1	1	1	1	1	1	0	1	FD
1	0	0	0	1	1	1	0	8E
d								

Registers

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

Flags Affected

Flags	s	z	-	h	-	p/v	-	c

- **s** set if result negative
- **z** set if result is 0
- **h** set if carry from bit 3
- **p/v** set if overflow
- **c** set if carry from bit 7

Opcode Matrix

	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
A	ADC A,A 1 4 8F	ADC A,B 1 4 88	ADC A,C 1 4 89	ADC A,D 1 4 8A	ADC A,E 1 4 8B	ADC A,H 1 4 8C	ADC A,L 1 4 8D	ADC A,(HL) 1 7 8E	ADC A,(IX+d) 3 19 DD8Enn	ADC A,(IY+d) 3 19 FD8Enn	ADC A,n 2 7 CEnn

Opcode Matrix Legend

Instruction
Size bytes — Cycle count
Opcode hex

Register | Memory | Implicit

3.2.2.2 - ADC 16 bit add with Carry

Addition with carry

7	6	5	4	3	2	1	0

$HL \longleftarrow HL + ss + Carry$

ADC HL, dd

1	1	1	0	1	1	0	1	ED
0	1	dd		1	0	1	0	

Registers

Value	dd
00	BC
01	DE
10	HL
11	SP

Flags Affected

Flags	s	z	-	h	-	p/v	-	c

- **s** set if result negative
- **z** set if result is 0
- **h** set if carry from bit 11
- **p/v** set if overflow
- **c** set if carry from bit 15

Opcode Matrix

	BC	DE	HL	SP
HL	ADC HL,BC 2 15 ED4A	ADC HL,DE 2 15 ED5A	ADC HL,HL 2 15 ED6A	ADC HL,SP 2 15 ED7A

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register

3.2.3 - SUB Subtract without Carry

Subtraction without Carry

$A \longleftarrow A - s$

This s operand is any of r, n, (HL), (IX+d), or (IY+d).

These possible op code/operand combinations are assembled as follows in the object code:

7	6	5	4	3	2	1	0

SUB r

1	0	0	1	0	r		

SUB n

1	1	0	1	0	1	1	0	D6
n								

SUB (HL)

1	0	0	1	0	1	1	0	96

SUB (IX+d)

1	1	0	1	1	1	0	1	DD
1	0	0	1	0	1	1	0	96
d								

SUB (IX+d)

1	1	1	1	1	1	0	1	FD
1	0	0	1	0	1	1	0	96
d								

Registers

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

Flags Affected

Flags	s	z	-	h	-	p/v	-	c

- **s** set if result negative
- **z** set if result is 0
- **h** set if borrow from bit 4
- **p/v** set if overflow
- **c** set if borrow

Opcode Matrix

	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
A	SUB A,A 1 4 97	SUB A,B 1 4 90	SUB A,C 1 4 91	SUB A,D 1 4 92	SUB A,E 1 4 93	SUB A,H 1 4 94	SUB A,L 1 4 95	SUB A,(HL) 1 7 96	SUB A,(IX+d) 3 19 DD96nn	SUB A,(IY+d) 3 19 FD96nn	SUB A,n 2 7 D6nn

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register	Memory	Implicit

3.2.4 - SBC Subtract with Carry

Subtraction with Carry

7	6	5	4	3	2	1	0

$A \longleftarrow A - r - Carry$

SBC A, r

1	0	0	1	1	r		

$A \longleftarrow A - n - Carry$

SBC A,n

1	1	0	1	1	1	1	0	DE

$A \longleftarrow A - (HL) - Carry$

SBC A, (HL)

1	0	0	1	1	1	1	0	9E

$A \longleftarrow A - (IX + d) - Carry$

SBC A, (IX+d)

1	1	0	1	1	1	0	1	DD
1	0	0	1	1	1	1	0	9E
d								

$A \longleftarrow A - (IY + d) - Carry$

SBC A, (IY+d)

1	1	1	1	1	1	0	1	FD
1	0	0	1	1	1	1	0	9E
d								

$A \longleftarrow A - ss - Carry$

SBC HL, ss

1	1	1	0	1	1	0	1	ED
0	1	dd		0	0	1	0	

Registers

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

Registers

Value	dd
00	BC
01	DE
10	HL
11	SP

Flags Affected

Flags	s	z	-	h	-	p/v	-	c

- **s** set if result negative
- **z** set if result is 0
- **h** set if borrow from bit 4
- **p/v** set if overflow
- **c** set if borrow

Opcode Matrix

	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n	BC	DE	HL	SP
A	SBC A,A 1 4 9F	SBC A,B 1 4 98	SBC A,C 1 4 99	SBC A,D 1 4 9A	SBC A,E 1 4 9B	SBC A,H 1 4 9C	SBC A,L 1 4 9D	SBC A,(HL) 1 7 9E	SBC A,(IX+d) 1 19 DD9Enn	SBC A,(IY+d) 1 19 FD9Enn	SBC A,n 2 7 DEnn				
HL												SBC HL,BC 2 15 ED42	SBC HL,DE 2 15 ED52	SBC HL,HL 2 15 ED62	SBC HL,SP 2 15 ED72

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register	Memory	Implicit

3.2.5 - AND

Binary AND

$A \longleftarrow A \wedge s$

7	6	5	4	3	2	1	0

AND r

1	0	1	0	0	r		

AND n

1	1	1	0	0	1	1	0	E6
n								

AND(HL)

1	0	1	0	0	1	1	0	A6

AND (IX+d)

1	1	0	1	1	1	0	1	DD
1	0	1	0	0	1	1	0	A6
d								

AND (IY+d)

1	1	1	1	1	1	0	1	FD
1	0	1	0	0	1	1	0	A6
d								

Registers

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

Flags Affected

Flags	s	z	-	h	-	p/v	-	c

s set if result negative

z set if result is 0

h set

p/v set if overflow

c reset

Opcode Matrix

	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
A	AND A,A 1 4 A7	AND A,B 1 4 A0	AND A,C 1 4 A1	AND A,D 1 4 A2	AND A,E 1 4 A3	AND A,H 1 4 A4	AND A,L 1 4 A5	AND A,(HL) 1 7 A6	AND A,(IX+d) 3 19 DDA6nn	AND A,(IY+d) 3 19 FDA6nn	AND A,n 2 7 E6nn

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register	Memory	Implicit

3.2.6 - OR

Binary OR

$A \longleftarrow A \vee s$

7	6	5	4	3	2	1	0	
OR r								
1	0	1	1	0	r			
OR n								
1	1	1	1	0	1	1	0	F6
n								
OR (HL)								
1	0	1	1	0	1	1	0	B6
OR (IX+d)								
1	1	0	1	1	1	0	1	DD
1	0	1	1	0	1	1	0	B6
d								
OR (IY+d)								
1	1	1	1	1	1	0	1	FD
1	0	1	1	0	1	1	0	B6
d								

Registers

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

Flags Affected

Flags	s	z	-	h	-	p/v	-	c

- **s** set if result negative
- **z** set if result is 0
- **h** reset
- **p/v** set if overflow
- **c** reset

Opcode Matrix

	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
A	OR A,A 1 4 B7	OR A,B 1 4 B0	OR A,C 1 4 B1	OR A,D 1 4 B2	OR A,E 1 4 B3	OR A,H 1 4 B4	OR A,L 1 4 B5	OR A,(HL) 1 7 B6	OR A,(IX+d) 3 19 DDB6nn	OR A,(IY+d) 3 19 FDB6nn	OR A,n 2 7 F6nn

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register	Memory	Implicit

3.2.7 - XOR

Binary Exclusive OR

$A \longleftarrow A \oplus s$

7	6	5	4	3	2	1	0

XOR r

1	0	1	0	1	r		

XOR n

1	1	1	0	1	1	1	0	EE
n								

XOR (HL)

1	0	1	0	1	1	1	0	AE

XOR (IX+d)

1	1	0	1	1	1	0	1	DD
1	0	1	0	1	1	1	0	AE
d								

XOR (IY+d)

1	1	1	1	1	1	0	1	FD
1	0	1	0	1	1	1	0	AE
d								

Registers

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

Flags Affected

Flags	s	z	-	h	-	p/v	-	c

s set if result negative

z set if result is 0

h reset

p/v set if overflow

c reset

Opcode Matrix

	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
A	XOR A,A 1 4 AF	XOR A,B 1 4 A8	XOR A,C 1 4 A9	XOR A,D 1 4 AA	XOR A,E 1 4 AB	XOR A,H 1 4 AC	XOR A,L 1 4 AD	XOR A,(HL) 1 7 AE	XOR A,(IX+d) 3 19 DDAEnn	XOR A,(IY+d) 3 19 FDAEnn	XOR A,n 2 7 EEnn

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register	Memory	Implicit

3.2.8 - INC Increment

Increment by 1

INC increments either an 8-bit register or an 16-bit register pair.

3.2.8.1 - INC 8-bit Increment

Increment 8-bit register by 1

7	6	5	4	3	2	1	0

$s \longleftarrow r + 1$

INC r

0	0	r			1	0	0

$(HL) \longleftarrow (HL) + 1$

INC (HL)

0	0	1	1	0	1	0	0	34

$(IX + d) \longleftarrow (IX + d) + 1$

INC (IX+d)

1	1	0	1	1	1	0	1	DD
0	0	1	1	0	1	0	0	34
d								

$(IY + d) \longleftarrow (IY + d) + 1$

INC (IY+d)

1	1	1	1	1	1	0	1	FD
0	0	1	1	0	1	0	0	34
d								

Registers

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

Flags Affected

Flags	s	z	-	h	-	p/v	-	-

s set if result negative

z set if result is 0

h set if carry from bit 3

p/v set if register was 0x7F before operation, reset otherwise

Opcode Matrix

	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
Op	INC A 1 4 3C	INC B 1 4 04	INC C 1 4 0C	INC D 1 4 14	INC E 1 4 1C	INC H 1 4 24	INC L 1 4 2C	INC (HL) 1 11 34	INC (IX+d) 3 23 DD34nn	INC (IY+d) 3 23 FD34nn

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register	Memory

3.2.8.2 - INC 16-bit Increment

Increment 16-bit register pair by 1

7	6	5	4	3	2	1	0

$dd \longleftarrow dd + 1$

INC qq

0	0	dd		0	0	1	1

$IX \longleftarrow IX + 1$

INC IX

1	1	0	1	1	1	0	1	DD
0	0	1	0	0	0	1	1	23

$IY \longleftarrow IY + 1$

INC IY

1	1	1	1	1	1	0	1	FD
0	0	1	0	0	0	1	1	23

Registers

Value	dd
00	BC
01	DE
10	HL
11	SP

Flags Affected

None.

Opcode Matrix

	BC	DE	HL	SP	IX	IY
Op	INC BC 1 6 03	INC DE 1 6 13	INC HL 1 6 23	INC SP 1 6 33	INC IX 2 10 DD23	INC IY 2 10 FD23

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register

3.2.9 - DEC Decrement

Decrement

DEC decrements either an 8-bit register or an 16-bit register pair.

3.2.9.1 - DEC 8-bit Decrement

Decrement

7	6	5	4	3	2	1	0

$r \longleftarrow r - 1$

DEC r

0	0	r			1	0	1

$(HL) \longleftarrow (HL) - 1$

DEC (HL)

0	0	1	1	0	1	0	1

$(IX + d) \longleftarrow (IX + d) - 1$

DEC (IX+d)

1	1	0	1	1	1	0	1	DD
0	0	1	1	0	1	0	1	35
d								

$(IY + d) \longleftarrow (IY + d) - 1$

DEC (IY+d)

1	1	1	1	1	1	0	1	FD
0	0	1	1	0	1	0	1	35
d								

Registers

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

Flags Affected

Flags	s	z	-	h	-	p/v	-	-

s set if result negative

z set if result is 0

h set if borrow from bit 4

p/v set if register was 0x80 before operation, reset otherwise

Opcode Matrix

	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
Op	DEC A 1 4 3D	DEC B 1 4 05	DEC C 1 4 0D	DEC D 1 4 15	DEC E 1 4 1D	DEC H 1 4 25	DEC L 1 4 2D	DEC (HL) 1 11 35	DEC (IX+d) 3 23 DD35nn	DEC (IY+d) 3 23 FD35nn

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register	Memory

3.2.9.2 - DEC 16-bit Decrement

Decrement 16-bit register pair

7	6	5	4	3	2	1	0

$dd \longleftarrow dd - 1$

DEC dd

0	0	dd	1	0	1	1

$IX \longleftarrow IX - 1$

DEC IX

1	1	0	1	1	1	0	1	DD
0	0	1	0	1	0	1	1	2B

$IY \longleftarrow IY - 1$

DEC IY

1	1	1	1	1	1	0	1	FD
0	0	1	0	1	0	1	1	2B

Registers

Value	dd
00	BC
01	DE
10	HL
11	SP

Flags Affected

None.

Opcode Matrix

	BC	DE	HL	SP	IX	IY
Op	DEC BC 1 6 0B	DEC DE 1 6 1B	DEC HL 1 6 2B	DEC SP 1 6 3B	DEC IX 2 10 DD2B	DEC IY 2 10 FD2B

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register

3.2.10 - CP

Comparison

$A - s$

7	6	5	4	3	2	1	0

CP r

1	0	1	1	1	r		

CP n

1	1	1	1	1	1	1	0	FE
n								

CP (HL)

1	0	1	1	1	1	1	0	BE

CP (IX+d)

1	1	0	1	1	1	0	1	DD
1	0	1	1	1	1	1	0	BE
d								

CP (IY+d)

1	1	1	1	1	1	0	1	FD
1	0	1	1	1	1	1	0	BE
d								

Registers

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

Flags Affected

Flags	s	z	-	h	-	p/v	-	c

- **s** set if result negative
- **z** set if result is 0
- **h** set if borrow from bit 4
- **p/v** set if overflow
- **c** set if borrow

Opcode Matrix

	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
Op	CP A 1 4 BF	CP B 1 4 B8	CP C 1 4 B9	CP D 1 4 BA	CP E 1 4 BB	CP H 1 4 BC	CP L 1 4 BD	CP (HL) 1 7 BE	CP (IX+d) 3 19 DDBEnn	CP (IY+d) 3 19 FDBEnn	CP n 2 7 FEnn

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register	Memory	Implicit

3.3 - Program Flow

Jump, Call and Return

3.3.1 - Jump absolute

7	6	5	4	3	2	1	0

$PC \longleftarrow nn$

JP nn

1	1	0	0	0	0	1	1	C3
7 15	nn						0 8	

$PC \longleftarrow nn\}$ if $ccc = true$

1	1	ccc	0	1	0	
7 15	nn				0 8	

$PC \longleftarrow HL$

JP (HL)

1	1	1	0	1	0	0	1	E9

$PC \longleftarrow IX$

JP (IX)

1	1	0	1	1	1	0	1	DD
1	1	1	0	1	0	0	1	E9

$PC \longleftarrow IY$

JP (IY)

1	1	1	1	1	1	0	1	FD
1	1	1	0	1	0	0	1	E9

Conditions

ccc	Abbrev	Condition	Flag
000	NZ	Non Zero	Z
001	Z	Zero	
010	NC	No Carry	C
011	C	Carry	
100	PO	Parity Odd	P/V
101	PE	Parity Even	
110	P	Sign Positive	S
111	M	Sign Negative	

Jumps to 16bit registers

Although the instruction `JP (HL)` looks like it's using indirect addressing, it doesn't. It takes the address in `HL` as the new `PC`, so it should be read as if it's `JP HL`.

The same applies for `JP (IX)` and `JP (IY)` - the actual register is used not the value at that address.

Flags Affected

None.

Opcode Matrix

	Uncond	C	NC	Z	NZ	PE	PO	N	P
JP nn	JP nn 3 10 C3nnnn	JP C,nn 3 10 DAnnnn	JP NC,nn 3 10 D2nnnn	JP Z,nn 3 10 CAnnnn	JP NZ,nn 3 10 C2nnnn	JP PE,nn 3 10 EAnnnn	JP PO,nn 3 10 E2nnnn	JP N,nn 3 10 FAnnnn	JP P,nn 3 10 F2nnnn
JP (HL)	JP (HL) 1 4 E9								
JP (IX)	JP (IX) 2 8 DDE9								
JP (IY)	JP (IY) 2 8 FDE9								

Opcode Matrix Legend

Size bytes	Instruction Opcode hex	Cycle count

Flow

3.3.2 - Jump Relative

7	6	5	4	3	2	1	0

$PC \longleftarrow PC + e$

JR e

0	0	0	1	1	0	0	0	18
e-2								

$PC \longleftarrow (PC) + e\}$ if $cc = true$

JR cc, e

0	0	1	cc	0	0	0	
e-2							

$B \longleftarrow B - 1$

$PC \longleftarrow PC + e\}$ if $B = 0$

DJNZ e

0	0	0	1	0	0	0	0	10
e-2								

Conditions

cc	Abbrev	Condition	Flag
00	NZ	Non Zero	Z
01	Z	Zero	
10	NC	No Carry	C
11	C	Carry	

Relative Jumps

For relative instructions the offset is taken from the address of the op code so is in the range -126 to 129. Assemblers usually account for the difference where the value in memory is e-2.

Timing

For JR then when a jump takes place then it takes 12(4,3,5) T-States whilst no jump 7(4,3) T-States.

For DJNZ if the jump takes place then it takes 13 (5,3,5) T-States. If no jump then 8 (5,3) T-States.

Flags Affected

None.

Opcode Matrix

	Uncond	C	NC	Z	NZ	B!=0
JR e	JR e 2 12 18nn	JR C,e 2 12 38nn	JR NC,e 2 12 30nn	JR Z,e 2 12 28nn	JR NZ,e 2 12 20nn	
DJNZ e						DJNZ e 2 13 10nn

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Flow

3.3.3 - Call subroutine

7	6	5	4	3	2	1	0

$(SP - 1) \longleftarrow PC_h$
$(SP - 2) \longleftarrow PC_l$
$SP \longleftarrow SP - 2$
$PC \longleftarrow nn$

CALL nn

1	1	0	0	1	1	0	1	CD
7							0	
15			nn				8	

$(SP - 1) \longleftarrow PC_h$
$(SP - 2) \longleftarrow PC_l$
$SP \longleftarrow SP - 2$
$PC \longleftarrow nn$
} if $ccc = true$

CALL ccc, nn

1	1	ccc		1	0	0	CD
7						0	
15			nn			8	

Conditions

ccc	Abbrev	Condition	Flag
000	NZ	Non Zero	Z
001	Z	Zero	
010	NC	No Carry	C
011	C	Carry	
100	PO	Parity Odd	P/V
101	PE	Parity Even	
110	P	Sign Positive	S
111	M	Sign Negative	

Timing

All call operation's take 17 (4,3,4,3,3) T-States, except for the conditional ones when the condition has not been met. In those instances it takes 10(4,3,3) T-States.

Flags Affected

None.

Opcode Matrix

	Uncond	C	NC	Z	NZ	PE	PO	N	P
CALL nn	CALL nn 3 17 CDnnnn	CALL C,nn 3 17 DCnnnn	CALL NC,nn 3 17 D4nnnn	CALL Z,nn 3 17 CCnnnn	CALL NZ,nn 3 17 C4nnnn	CALL PE,nn 3 17 ECnnnn	CALL PO,nn 3 17 E4nnnn	CALL N,nn 3 17 FCnnnn	CALL P,nn 3 17 F4nnnn

Opcode Matrix Legend

Instruction Size bytes Opcode hex Cycle count	Flow

3.3.4 - Return from Subroutine

7	6	5	4	3	2	1	0

$PC_l \longleftarrow (SP)$
$PC_h \longleftarrow (SP + 1)$
$SP \longleftarrow SP + 2$

RET

1	1	0	0	1	0	0	1

$PC_l \longleftarrow (SP)$
$PC_h \longleftarrow (SP + 1)$ } if $ccc = true$
$SP \longleftarrow SP + 2$

RET ccc

1	1	ccc	0	0	0

Conditions

ccc	Abbrev	Condition	Flag
000	NZ	Non Zero	Z
001	Z	Zero	
010	NC	No Carry	C
011	C	Carry	
100	PO	Parity Odd	P/V
101	PE	Parity Even	
110	P	Sign Positive	S
111	M	Sign Negative	

Timing

The unconditional RET takes 10 (4,3,3) T-States. The conditional RET takes 17(5,3,3) T-States if the condition is true and 5 T-States if false and no return was performed.

Flags Affected

None.

Opcode Matrix

	Uncond	C	NC	Z	NZ	PE	PO	N	P
RET	RET 1 10 C9	RET C 1 11 D8	RET NC 1 11 D0	RET Z 1 11 C8	RET NZ 1 11 C0	RET PE 1 11 E8	RET PO 1 11 E0	RET N 1 11 F8	RET P 1 11 F0

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Flow

3.3.5 - RST

Invoke a Reset

RST performs a reset. Specifically it calls a routine at one of 8 addresses at the base of memory. It is the equivalent of performing a CALL to that address except the RST instruction is just 1 byte compared to 3 for CALL and is slightly faster.

$(SP - 1) \longleftarrow PC_h$
$(SP - 2) \longleftarrow PC_l$
$SP \longleftarrow SP - 2$
$PC_h \longleftarrow 0$
$PC_l \longleftarrow b * 8$

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

Bits

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

Issues with RST instructions

Assemblers use different conventions for the RST instruction. Some use numbers 0...7 whilst others use the address of the code invoked. They are all equivalent, as there are just 8 possible instruction codes.

Address	OP Code	RST Instruction			Action
0000	C7	RST 0			Reset machine
0008	CF	RST 1	RST 8		Operating System Specific
0010	D7	RST 2	RST $10	RST 16	
0018	DF	RST 3	RST $18	RST 24	
0020	E7	RST 4	RST $20	RST 32	
0028	EF	RST 5	RST $28	RST 40	
0030	F7	RST 6	RST $30	RST 48	
0038	FF	RST 7	RST $38	RST 56	Interrupt Handler in Mode 1

Flags Affected

None.

Opcode Matrix

	Reset routine							
	0	1	2	3	4	5	6	7
RST	RST 0 1 11 C7	RST 1 1 11 CF	RST 2 1 11 D7	RST 3 1 11 DF	RST 4 1 11 E7	RST 5 1 11 EF	RST 6 1 11 F7	RST 7 1 11 FF

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Special

3.3.6 - Return from Interrupt

7	6	5	4	3	2	1	0

$PC_l \longleftarrow (SP)$
$PC_h \longleftarrow (SP + 1)$
$SP \longleftarrow SP + 2$

RETI

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

$PC_l \longleftarrow (SP)$
$PC_h \longleftarrow (SP + 1)$
$SP \longleftarrow SP + 2$
$IFF_1 \longleftarrow IFF_2$

RETN

1	1	1	0	1	1	0	1	ED
0	1	0	0	0	1	0	1	45

Flags Affected

None.

Opcode Matrix

	RETI	RETN
Op	RETI 2 14 ED4D	RETN 2 14 ED45

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Interrupt

3.4 - Stack

Push Pull onto the stack

7	6	5	4	3	2	1	0

$(SP-2) \longleftarrow qq_l, (SP-1) \longleftarrow qq_h$

PUSH qq

1	1	qq	0	1	0	1	

$(SP-2) \longleftarrow IX_l, (SP-1) \longleftarrow IX_h$

PUSH IX

1	1	0	1	1	1	0	1	DD
1	1	1	0	0	1	0	1	E5

$(SP-2) \longleftarrow IY_l, (SP-1) \longleftarrow IY_h$

PUSH IY

1	1	1	1	1	1	0	1	FD
1	1	1	0	0	1	0	1	E5

$qq_h \longleftarrow (SP-1), qq_l \longleftarrow (SP)$

POP qq

1	1	qq	0	0	0	1	

$IX_h \longleftarrow (SP-1), IX_l \longleftarrow (SP)$

POP IX

1	1	0	1	1	1	0	1	DD
1	1	1	0	0	0	0	1	E1

$IY_h \longleftarrow (SP-1), IY_l \longleftarrow (SP)$

POP IY

1	1	1	1	1	1	0	1	FD
1	1	1	0	0	0	0	1	E1

Registers

Value	qq
00	BC
01	DE
10	HL
11	AF

Flags Affected

None.

Opcode Matrix

	AF	BC	DE	HL	IX	IY
PUSH	PUSH AF 1 11 F5	PUSH BC 1 11 C5	PUSH DE 1 11 D5	PUSH HL 1 11 E5	PUSH IX 2 15 DDE5	PUSH IY 2 15 FDE5
POP	POP AF 1 10 F1	POP BC 1 10 C1	POP DE 1 10 D1	POP HL 1 10 E1	POP IX 2 14 DDE1	POP IY 2 14 FDE1

Opcode Matrix Legend

Size bytes	Instruction Opcode hex	Cycle count

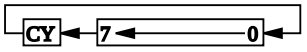
Memory

3.5 - Rotate and Shift

Rotate Shift instructions

3.5.1 - RL Rotate bits left with Carry

Rotate bits left with carry



7	6	5	4	3	2	1	0

RLA

0	0	0	1	0	1	1	1	17

RL r

1	1	0	0	1	0	1	1	CB
0	0	0	1	0	r			

RL (HL)

1	1	0	0	1	0	1	1	CB
0	0	0	1	0	1	1	0	16

RL (IX+d)

1	1	0	1	1	1	0	1	DD
1	1	0	0	1	0	1	1	CB
d								
0	0	0	1	0	1	1	0	16

RL (IY+d)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
0	0	0	1	0	1	1	0	16

Registers

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

Flags Affected

Flags	s	z	-	h	-	p/v	-	c

- **s** set if result negative
- **z** set if result is 0
- **h** reset
- **p/v** set if parity even, reset if parity odd
- **c** data from bit 7 of source register

Opcode Matrix

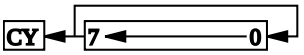
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
RL	RLA 1 4 17									
RL	RL A 2 8 CB17	RL B 2 8 CB10	RL C 2 8 CB11	RL D 2 8 CB12	RL E 2 8 CB13	RL H 2 8 CB14	RL L 2 8 CB15	RL (HL) 2 15 CB16	RL (IX+d) 4 23 DDCBnn16	RL (IY+d) 4 23 FDCBnn16

Opcode Matrix Legend

Instruction / Size bytes / Opcode hex / Cycle count	Register	Memory

3.5.2 - RLC Rotate bits left with Carry

Rotate bits left with carry



7	6	5	4	3	2	1	0

RLCA

0	0	0	0	0	1	1	1	07

RLC r

1	1	0	0	1	0	1	1	CB
0	0	0	0	0	r			

RLC (HL)

1	1	0	0	1	0	1	1	CB
0	0	0	0	0	1	1	0	06

RLC (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	0	1	1	0	06

RLC (IY+d)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
0	0	0	0	0	1	1	0	06

Registers

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

Flags Affected

Flags	s	z	-	h	-	p/v	-	c

s set if result negative

z set if result is 0

h reset

p/v set if parity even, reset if parity odd

c data from bit 7 of source register

Opcode Matrix

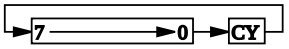
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
RLC	RLCA 1 4 07									
RLC	RLC A 2 8 CB07	RLC B 2 8 CB00	RLC C 2 8 CB01	RLC D 2 8 CB02	RLC E 2 8 CB03	RLC H 2 8 CB04	RLC L 2 8 CB05	RLC (HL) 2 8 CB06	RLC (IX+d) 4 23 DDCBnn06	RLC (IY+d) 4 23 FDCBnn06

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register	Memory

3.5.3 - RR Rotate bits right with Carry

Rotate bits right with carry



7	6	5	4	3	2	1	0

RRA

0	0	0	1	1	1	1	1	1F

RR r

1	1	0	0	1	0	1	1	CB
0	0	0	1	1	r			

RR (HL)

1	1	0	0	1	0	1	1	CB
0	0	0	1	1	1	1	0	1E

RR(IX+d)

1	1	0	1	1	1	0	1	DD
1	1	0	0	1	0	1	1	CB
d								
0	0	0	1	1	1	1	0	1E

RR (IY+d)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
0	0	0	1	1	1	1	0	1E

Registers

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

Flags Affected

Flags	s	z	-	h	-	p/v	-	c

s set if result negative

z set if result is 0

h reset

p/v set if parity even, reset if parity odd

c data from bit 0 of source register

Opcode Matrix

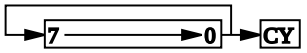
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
RR	RRA 1 4 1F									
RR	RR A 2 8 CB1F	RR B 2 8 CB18	RR C 2 8 CB19	RR D 2 8 CB1A	RR E 2 8 CB1B	RR H 2 8 CB1C	RR L 2 8 CB1D	RR (HL) 2 15 CB1E	RR (IX+d) 4 23 DDCBnn1E	RR (IY+d) 4 23 FDCBnn1E

Opcode Matrix Legend

Size bytes	Instruction / Opcode hex	Cycle count	Register	Memory

3.5.4 - RRC Rotate bits right with Carry

Rotate bits left with carry



7	6	5	4	3	2	1	0

RRCA

0	0	0	0	1	1	1	1	0F

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)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
0	0	0	0	1	1	1	0	0E

Registers

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

Flags Affected

Flags	s	z	-	h	-	p/v	-	c

- **s** set if result negative
- **z** set if result is 0
- **h** reset
- **p/v** set if parity even, reset if parity odd
- **c** data from bit 0 of source register

Opcode Matrix

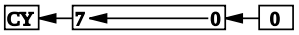
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
RRC	RRCA 1 4 0F									
RRC	RRC A 2 8 CB0F	RRC B 2 8 CB08	RRC C 2 8 CB09	RRC D 2 8 CB0A	RRC E 2 8 CB0B	RRC H 2 8 CB0C	RRC L 2 8 CB0D	RRC (HL) 2 15 CB0E	RRC (IX+d) 4 23 DDCBnn0E	RRC (IY+d) 4 23 FDCBnn0E

Opcode Matrix Legend

Instruction (Size bytes / Opcode hex / Cycle count)	Register	Memory

3.5.5 - SLA Shift bits left with Carry

Shift bits left with carry



7	6	5	4	3	2	1	0

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)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
0	0	1	0	0	1	1	0	26

Registers

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

Flags Affected

Flags	s	z	-	h	-	p/v	-	c

- **s** set if result negative
- **z** set if result is 0
- **h** reset
- **p/v** set if parity even, reset if parity odd
- **c** data from bit 7 of source register

Opcode Matrix

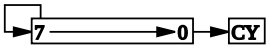
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
SLA	SLA A 2 8 CB27	SLA B 2 8 CB20	SLA C 2 8 CB21	SLA D 2 8 CB22	SLA E 2 8 CB23	SLA H 2 8 CB24	SLA L 2 8 CB25	SLA (HL) 2 15 CB26	SLA (IX+d) 4 23 DDCBnn26	SLA (IY+d) 4 23 FDCBnn26

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register	Memory

3.5.6 - SRA Rotate bits right with Carry

Rotate bits right with carry, bit 7 remains unchanged



An arithmetic shift right 1 bit position is performed on the contents of operand. The contents of bit 0 are copied to the Carry flag and the previous contents of bit 7 remain unchanged.

Registers

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

7	6	5	4	3	2	1	0

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)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
0	0	1	0	1	1	1	0	2E

Flags Affected

Flags	s	z	-	h	-	p/v	-	c

- **s** set if result negative
- **z** set if result is 0
- **h** reset
- **p/v** set if parity even, reset if parity odd
- **c** data from bit 0 of source register

Opcode Matrix

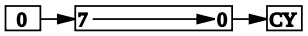
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
SRA	SRA A 2 8 CB2F	SRA B 2 8 CB28	SRA C 2 8 CB29	SRA D 2 8 CB2A	SRA E 2 8 CB2B	SRA H 2 8 CB2C	SRA L 2 8 CB2D	SRA (HL) 2 15 CB2E	SRA (IX+d) 4 23 DDCBnn2E	SRA (IY+d) 4 23 FDCBnn2E

Opcode Matrix Legend

Size bytes	Instruction Opcode hex	Cycle count	Register	Memory

3.5.7 - SRL Rotate bits right with Carry

Rotate bits right with carry, bit 7 is reset



7	6	5	4	3	2	1	0

SRL r

1	1	0	0	1	0	1	1	CB
0	0	1	1	1	r			

SRL (HL)

1	1	0	0	1	0	1	1	CB
0	0	1	1	1	1	1	0	3E

SRL (IX+d)

1	1	0	1	1	1	0	1	DD
1	1	0	0	1	0	1	1	CB
d								
0	0	1	1	1	1	1	0	3E

SRL (IY+d)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
0	0	1	1	1	1	1	0	3E

Registers

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

Flags Affected

Flags	s	z	-	h	-	p/v	-	c

s set if result negative
z set if result is 0
h reset
p/v set if parity even, reset if parity odd
c data from bit 0 of source register

Opcode Matrix

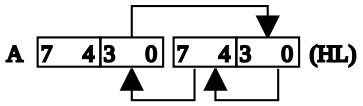
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
SRL	SRL A 2 8 CB3F	SRL B 2 8 CB38	SRL C 2 8 CB39	SRL D 2 8 CB3A	SRL E 2 8 CB3B	SRL H 2 8 CB3C	SRL L 2 8 CB3D	SRL (HL) 2 15 CB3E	SRL (IX+d) 4 23 DDCBnn3E	SRL (IY+d) 4 23 FDCBnn3E

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register	Memory

3.5.8 - RLD

Rotate bit pairs in A and (HL) left



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

The contents of the low-order four bits (bits 3, 2, 1, and 0) of the memory location (HL) are copied to 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 to 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 to the low-order four bits of memory location (HL). The contents of the high-order bits of the Accumulator are unaffected.

Flags Affected

Flags	s	z	-	h	-	p/v	-	-

s set if result negative
z set if result is 0
h reset
p/v set if parity even, reset if parity odd

Opcode Matrix

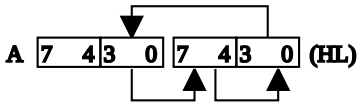
	(HL)
Op	RLD (HL) 2 18 ED6F

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Memory

3.5.9 - RRD

Rotate bit pairs in A and (HL) right



7	6	5	4	3	2	1	0	
1	1	1	0	1	1	0	1	ED
0	1	1	0	0	1	1	1	67

The contents of the low-order four bits (bits 3, 2, 1, and 0) of memory location (HL) are copied to the low-order four bits of the Accumulator (Register A). The previous contents of the low-order four bits of the Accumulator are copied to 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 to the low-order four bits of (HL). The contents of the high-order bits of the Accumulator are unaffected.

Flags Affected

Flags	s	z	-	h	-	p/v	-	-

s set if result negative
z set if result is 0
h reset
p/v set if parity even, reset if parity odd

Opcode Matrix

	(HL)
Op	RRD (HL) 2 18 ED67

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Memory

3.6 - Bit Manipulation

Bit Manipulation instructions

3.6.1 - BIT

Test if a specific bit is set

7	6	5	4	3	2	1	0

$Z \longleftarrow \overline{r_b}$

BIT b, r

1	1	0	0	1	0	1	1	CB
0	1	b			r			

$Z \longleftarrow \overline{(HL)_b}$

BIT b, (HL)

1	1	0	0	1	0	1	1	CB
0	1	b			1	1	0	

$Z \longleftarrow \overline{(IX + d)_b}$

BIT b, (IX+d)

1	1	0	1	1	1	0	1	DD
1	1	0	0	1	0	1	1	CB
d								
0	1	b			1	1	0	

$Z \longleftarrow \overline{(IY + d)_b}$

BIT b, (IY+d)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
0	1	b			1	1	0	

Registers

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

Bits

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

Z is set if the specified bit in the source is 0, otherwise it is cleared.

Flags Affected

Flags	-	z	-	h	-	-	n	-

z set if the specified bit is 0

h set

n reset

Opcode Matrix

	Source									
	A	**B**	**C**	**D**	**E**	**H**	**L**	**(HL)**	**(IX+d)**	**(IY+d)**
BIT 0	BIT 0,A 2 8 CB47	BIT 0,B 2 8 CB40	BIT 0,C 2 8 CB41	BIT 0,D 2 8 CB42	BIT 0,E 2 8 CB43	BIT 0,H 2 8 CB44	BIT 0,L 2 8 CB45	BIT 0,(HL) 2 12 CB46	BIT 0,(IX+d) 4 20 DDCBnn46	BIT 0,(IY+d) 4 20 FDCBnn46
BIT 1	BIT 1,A 2 8 CB4F	BIT 1,B 2 8 CB48	BIT 1,C 2 8 CB49	BIT 1,D 2 8 CB4A	BIT 1,E 2 8 CB4B	BIT 1,H 2 8 CB4C	BIT 1,L 2 8 CB4D	BIT 1,(HL) 2 12 CB4E	BIT 1,(IX+d) 4 20 DDCBnn4E	BIT 1,(IY+d) 4 20 FDCBnn4E
BIT 2	BIT 2,A 2 8 CB57	BIT 2,B 2 8 CB50	BIT 2,C 2 8 CB51	BIT 2,D 2 8 CB52	BIT 2,E 2 8 CB53	BIT 2,H 2 8 CB54	BIT 2,L 2 8 CB55	BIT 2,(HL) 2 12 CB56	BIT 2,(IX+d) 4 20 DDCBnn56	BIT 2,(IY+d) 4 20 FDCBnn56
BIT 3	BIT 3,A 2 8 CB5F	BIT 3,B 2 8 CB58	BIT 3,C 2 8 CB59	BIT 3,D 2 8 CB5A	BIT 3,E 2 8 CB5B	BIT 3,H 2 8 CB5C	BIT 3,L 2 8 CB5D	BIT 3,(HL) 2 12 CB5E	BIT 3,(IX+d) 4 20 DDCBnn5E	BIT 3,(IY+d) 4 20 FDCBnn5E
BIT 4	BIT 4,A 2 8 CB67	BIT 4,B 2 8 CB60	BIT 4,C 2 8 CB61	BIT 4,D 2 8 CB62	BIT 4,E 2 8 CB63	BIT 4,H 2 8 CB64	BIT 4,L 2 8 CB65	BIT 4,(HL) 2 12 CB66	BIT 4,(IX+d) 4 20 DDCBnn66	BIT 4,(IY+d) 4 20 FDCBnn66
BIT 5	BIT 5,A 2 8 CB6F	BIT 5,B 2 8 CB68	BIT 5,C 2 8 CB69	BIT 5,D 2 8 CB6A	BIT 5,E 2 8 CB6B	BIT 5,H 2 8 CB6C	BIT 5,L 2 8 CB6D	BIT 5,(HL) 2 12 CB6E	BIT 5,(IX+d) 4 20 DDCBnn6E	BIT 5,(IY+d) 4 20 FDCBnn6E
BIT 6	BIT 6,A 2 8 CB77	BIT 6,B 2 8 CB70	BIT 6,C 2 8 CB71	BIT 6,D 2 8 CB72	BIT 6,E 2 8 CB73	BIT 6,H 2 8 CB74	BIT 6,L 2 8 CB75	BIT 6,(HL) 2 12 CB76	BIT 6,(IX+d) 4 20 DDCBnn76	BIT 6,(IY+d) 4 20 FDCBnn76
BIT 7	BIT 7,A 2 8 CB7F	BIT 7,B 2 8 CB78	BIT 7,C 2 8 CB79	BIT 7,D 2 8 CB7A	BIT 7,E 2 8 CB7B	BIT 7,H 2 8 CB7C	BIT 7,L 2 8 CB7D	BIT 7,(HL) 2 12 CB7E	BIT 7,(IX+d) 4 20 DDCBnn7E	BIT 7,(IY+d) 4 20 FDCBnn7E

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register	Memory

3.6.2 - RES

Reset a specific bit

7	6	5	4	3	2	1	0

$r_b \longleftarrow 0$

RES b, r

1	1	0	0	1	0	1	1	CB
1	0	b			r			

$(HL)_b \longleftarrow 0$

RES b, (HL)

1	1	0	0	1	0	1	1	CB
1	0	b			1	1	0	

$(IX + d)_b \longleftarrow 0$

RES b, (IX+d)

1	1	0	1	1	1	0	1	DD
1	1	0	0	1	0	1	1	CB
d								
1	0	b			1	1	0	

$(IY + d)_b \longleftarrow 0$

RES b, (IY+d)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
1	0	b			1	1	0	

Registers

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

Bits

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

Z is set if the specified bit in the source is 0, otherwise it is cleared.

Flags Affected

None.

Opcode Matrix

	Source									
	A	**B**	**C**	**D**	**E**	**H**	**L**	**(HL)**	**(IX+d)**	**(IY+d)**
RES 0	RES 0,A 2 8 CB87	RES 0,B 2 8 CB80	RES 0,C 2 8 CB81	RES 0,D 2 8 CB82	RES 0,E 2 8 CB83	RES 0,H 2 8 CB84	RES 0,L 2 8 CB85	RES 0,(HL) 2 15 CB86	RES 0,(IX+d) 4 23 DDCBnn86	RES 0,(IY+d) 4 23 FDCBnn86
RES 1	RES 1,A 2 8 CB8F	RES 1,B 2 8 CB88	RES 1,C 2 8 CB89	RES 1,D 2 8 CB8A	RES 1,E 2 8 CB8B	RES 1,H 2 8 CB8C	RES 1,L 2 8 CB8D	RES 1,(HL) 2 15 CB8E	RES 1,(IX+d) 4 23 DDCBnn8E	RES 1,(IY+d) 4 23 FDCBnn8E
RES 2	RES 2,A 2 8 CB97	RES 2,B 2 8 CB90	RES 2,C 2 8 CB91	RES 2,D 2 8 CB92	RES 2,E 2 8 CB93	RES 2,H 2 8 CB94	RES 2,L 2 8 CB95	RES 2,(HL) 2 15 CB96	RES 2,(IX+d) 4 23 DDCBnn96	RES 2,(IY+d) 4 23 FDCBnn96
RES 3	RES 3,A 2 8 CB9F	RES 3,B 2 8 CB98	RES 3,C 2 8 CB99	RES 3,D 2 8 CB9A	RES 3,E 2 8 CB9B	RES 3,H 2 8 CB9C	RES 3,L 2 8 CB9D	RES 3,(HL) 2 15 CB9E	RES 3,(IX+d) 4 23 DDCBnn9E	RES 3,(IY+d) 4 23 FDCBnn9E
RES 4	RES 4,A 2 8 CBA7	RES 4,B 2 8 CBA0	RES 4,C 2 8 CBA1	RES 4,D 2 8 CBA2	RES 4,E 2 8 CBA3	RES 4,H 2 8 CBA4	RES 4,L 2 8 CBA5	RES 4,(HL) 2 15 CBA6	RES 4,(IX+d) 4 23 DDCBnnA6	RES 4,(IY+d) 4 23 FDCBnnA6
RES 5	RES 5,A 2 8 CBAF	RES 5,B 2 8 CBA8	RES 5,C 2 8 CBA9	RES 5,D 2 8 CBAA	RES 5,E 2 8 CBAB	RES 5,H 2 8 CBAC	RES 5,L 2 8 CBAD	RES 5,(HL) 2 15 CBAE	RES 5,(IX+d) 4 23 DDCBnnAE	RES 5,(IY+d) 4 23 FDCBnnAE
RES 6	RES 6,A 2 8 CBB7	RES 6,B 2 8 CBB0	RES 6,C 2 8 CBB1	RES 6,D 2 8 CBB2	RES 6,E 2 8 CBB3	RES 6,H 2 8 CBB4	RES 6,L 2 8 CBB5	RES 6,(HL) 2 15 CBB6	RES 6,(IX+d) 4 23 DDCBnnB6	RES 6,(IY+d) 4 23 FDCBnnB6
RES 7	RES 7,A 2 8 CBBF	RES 7,B 2 8 CBB8	RES 7,C 2 8 CBB9	RES 7,D 2 8 CBBA	RES 7,E 2 8 CBBB	RES 7,H 2 8 CBBC	RES 7,L 2 8 CBBD	RES 7,(HL) 2 15 CBBE	RES 7,(IX+d) 4 23 DDCBnnBE	RES 7,(IY+d) 4 23 FDCBnnBE

Opcode Matrix Legend

Instruction
Size bytes | Cycle count
Opcode hex

Register | Memory

3.6.3 - SET

Set a specific bit

7	6	5	4	3	2	1	0

$r_b \longleftarrow 1$

SET b, r

1	1	0	0	1	0	1	1	CB
1	1	b			r			

$(HL)_b \longleftarrow 1$

SET b, (HL)

1	1	0	0	1	0	1	1	CB
1	1	b			1	1	0	

$(IX + d)_b \longleftarrow 1$

SET b, (IX+d)

1	1	0	1	1	1	0	1	DD
1	1	0	0	1	0	1	1	CB
d								
1	1	b			1	1	0	

$(IY + d)_b \longleftarrow 1$

SET b, (IY+d)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
1	1	b			1	1	0	

Registers

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

Bits

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

Z is set if the specified bit in the source is 0, otherwise it is cleared.

Flags Affected

None.

Opcode Matrix

	Source									
	A	**B**	**C**	**D**	**E**	**H**	**L**	**(HL)**	**(IX+d)**	**(IY+d)**
SET 0	SET 0,A 2 8 CBC7	SET 0,B 2 8 CBC0	SET 0,C 2 8 CBC1	SET 0,D 2 8 CBC2	SET 0,E 2 8 CBC3	SET 0,H 2 8 CBC4	SET 0,L 2 8 CBC5	SET 0,(HL) 2 15 CBC6	SET 0,(IX+d) 4 23 DDCBnnC6	SET 0,(IY+d) 4 23 FDCBnnC6
SET 1	SET 1,A 2 8 CBCF	SET 1,B 2 8 CBC8	SET 1,C 2 8 CBC9	SET 1,D 2 8 CBCA	SET 1,E 2 8 CBCB	SET 1,H 2 8 CBCC	SET 1,L 2 8 CBCD	SET 1,(HL) 2 15 CBCE	SET 1,(IX+d) 4 23 DDCBnnCE	SET 1,(IY+d) 4 23 FDCBnnCE
SET 2	SET 2,A 2 8 CBD7	SET 2,B 2 8 CBD0	SET 2,C 2 8 CBD1	SET 2,D 2 8 CBD2	SET 2,E 2 8 CBD3	SET 2,H 2 8 CBD4	SET 2,L 2 8 CBD5	SET 2,(HL) 2 15 CBD6	SET 2,(IX+d) 4 23 DDCBnnD6	SET 2,(IY+d) 4 23 FDCBnnD6
SET 3	SET 3,A 2 8 CBDF	SET 3,B 2 8 CBD8	SET 3,C 2 8 CBD9	SET 3,D 2 8 CBDA	SET 3,E 2 8 CBDB	SET 3,H 2 8 CBDC	SET 3,L 2 8 CBDD	SET 3,(HL) 2 15 CBDE	SET 3,(IX+d) 4 23 DDCBnnDE	SET 3,(IY+d) 4 23 FDCBnnDE
SET 4	SET 4,A 2 8 CBE7	SET 4,B 2 8 CBE0	SET 4,C 2 8 CBE1	SET 4,D 2 8 CBE2	SET 4,E 2 8 CBE3	SET 4,H 2 8 CBE4	SET 4,L 2 8 CBE5	SET 4,(HL) 2 15 CBE6	SET 4,(IX+d) 4 23 DDCBnnE6	SET 4,(IY+d) 4 23 FDCBnnE6
SET 5	SET 5,A 2 8 CBEF	SET 5,B 2 8 CBE8	SET 5,C 2 8 CBE9	SET 5,D 2 8 CBEA	SET 5,E 2 8 CBEB	SET 5,H 2 8 CBEC	SET 5,L 2 8 CBED	SET 5,(HL) 2 15 CBEE	SET 5,(IX+d) 4 23 DDCBnnEE	SET 5,(IY+d) 4 23 FDCBnnEE
SET 6	SET 6,A 2 8 CBF7	SET 6,B 2 8 CBF0	SET 6,C 2 8 CBF1	SET 6,D 2 8 CBF2	SET 6,E 2 8 CBF3	SET 6,H 2 8 CBF4	SET 6,L 2 8 CBF5	SET 6,(HL) 2 15 CBF6	SET 6,(IX+d) 4 23 DDCBnnF6	SET 6,(IY+d) 4 23 FDCBnnF6
SET 7	SET 7,A 2 8 CBFF	SET 7,B 2 8 CBF8	SET 7,C 2 8 CBF9	SET 7,D 2 8 CBFA	SET 7,E 2 8 CBFB	SET 7,H 2 8 CBFC	SET 7,L 2 8 CBFD	SET 7,(HL) 2 15 CBFE	SET 7,(IX+d) 4 23 DDCBnnFE	SET 7,(IY+d) 4 23 FDCBnnFE

Opcode Matrix Legend

Instruction
Size bytes — Cycle count
Opcode hex

Register | Memory

3.7 - Exchanges

Exchange registers

These instructions exchange values between registers.

7	6	5	4	3	2	1	0

$AF \longleftrightarrow AF'$

EX AF, AF'

0	0	0	0	1	0	0	0	08

$BC \longleftrightarrow BC', DE \longleftrightarrow DE', HL \longleftrightarrow HL'$

EXX

1	1	0	1	1	0	0	1	D9

$DE \longleftrightarrow HL$

EX DE, HL

1	1	1	0	1	0	1	1	EB

$H \longleftrightarrow (SP+1), L \longleftrightarrow (SP)$

EX (SP), HL

1	1	1	0	0	0	1	1	E3

$IX_h \longleftrightarrow (SP+1), IX_l \longleftrightarrow (SP)$

EX (SP), IX

1	1	0	1	1	1	0	1	DD
1	1	1	0	0	0	1	1	E3

$IY_h \longleftrightarrow (SP+1), IY_l \longleftrightarrow (SP)$

EX (SP), IY

1	1	1	1	1	1	0	1	FD
1	1	1	0	0	0	1	1	E3

`EX AF, AF'` (0x08) allows the programmer to switch between the two pairs of Accumulator flag registers.

`EX DE, HL` (0xEB) exchanges the values between those two registers.

`EXX` (0xD9) allows the programmer to switch BC, DE and HL and BC', DE' and HL' register pairs.

Internally these instructions toggles a flip-flop which determines which register or register set is the active one. This minimises how long the instruction takes as no data is transferred - just a single bit changes state.

`EX (SP),HL` exchanges HL with the last value pushed on the stack.

Flags Affected

None.

Opcode Matrix

	AF'	HL	IX	IY	BC',DE',HL'
AF	EX AF, AF' 1 4 08				
DE		EX DE, HL 1 4 EB			
(SP)		EX (SP), HL 1 19 E3	EX (SP), IX 2 23 DDE3	EX (SP), IY 2 23 FDE3	
BC,DE,HL					EXX 1 4 D9

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register	Memory

3.8 - Block Copy or Search of memory

Copy or search block of memory

3.8.1 - Block Copy

Copy block of memory

The Block copy instructions allow for data to be moved around in memory. The programmer needs to configure the 16 bit registers to define the properties of the move: HL is the source address to copy from; DE is the destination address to copy to; BC is the number of bytes to copy.

$$\left.\begin{array}{l} HL \longleftarrow HL + 1 \\ DE \longleftarrow DE + 1 \end{array}\right\} \text{ if } D = 0$$
$$\left.\begin{array}{l} HL \longleftarrow HL - 1 \\ DE \longleftarrow DE - 1 \end{array}\right\} \text{ if } D = 1$$
$$BC \longleftarrow BC - 1$$

$$\left.\begin{array}{l} \\ \\ \\ \\ \\ \end{array}\right\} \text{ repeat while } \left\{\begin{array}{l} L = 1 \\ BC = 0 \end{array}\right.$$

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	0	1	ED
1	0	1	L	D	0	0	0	

D 0=Increment, 1=Decrement HL after each iteration.

L If set then if $BC = 0$ at the end if the instruction then $PC \longleftarrow PC - 2$ so that the instruction is repeated.
If BC=0 at start of a repeatable instruction then 65536 iterations will occur.

The LD* instructions then perform the equivalent of the following:

1. Copy a byte of memory from (HL) to (DE)
2. Decrement BC by one
3. HL and DE are either incremented (for LDI/LDIR) or decremented (for LDD/LDDR) by one.
4. The LDIR and LDDR instructions will loop back to step one if $BC = 0$

Timing

For the non-repeating instructions, they take 16(4,4,3,5) T-States to execute.

For the repeating instructions, they take either 21(4,4,3,5,5) T-States when they loop and 16(4,4,3,5) T-States when terminating.

Also note, that for these instructions the timing is for each iteration, not for the entire run. So if LDIR is run with BC=4 then the number of T-States for the entire operation would take 79(21+21+21+16) T-States.

Flags Affected

Flags	-	-	-	h	-	p/v	-	-

h Reset

p/v Non-repeating: Set if BC-1 != 0, otherwise reset
Repeating: N/A as BC=0 after instruction completes

Opcode Matrix

	Increment	Decrement
Single Copy	LDI 2 16 EDA0	LDD 2 16 EDA8
Repeat Copy	LDIR 2 21 EDB0	LDDR 2 21 EDB8

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Memory

3.8.2 - Block Search of memory

Search block of memory

The Block compare instructions allow for data to be searched for in memory. The programmer needs to configure the following registers to define the properties of the search: HL is the source address to search from; BC is the number of bytes to search. A is set to the value to search for.

$$\left.\begin{array}{l} A - (HL) \\ HL \longleftarrow HL + 1 \text{ if } D = 0 \\ HL \longleftarrow HL - 1 \text{ if } D = 1 \\ BC \longleftarrow BC - 1 \end{array}\right\} \text{ repeat while } \left\{\begin{array}{l} L = 1 \\ A = (HL) \\ BC = 0 \end{array}\right.$$

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	0	1	ED
1	0	1	L	D	0	0	1	

D 0=Increment, 1=Decrement HL after each iteration.

L If set then if $BC = 0$ at the end if the instruction then $PC \longleftarrow PC - 2$ so that the instruction is repeated.
If BC=0 at start of a repeatable instruction then 65536 iterations will occur.

The CP* instructions compare memory against the Accumulator

1. Calculate difference between A and content of memory in (HL) to set/clear Z flag
2. Decrement BC by one
3. HL is either incremented (for CPI/CPIR) or decremented (for CPD/CPDR) by one.
4. The CPIR and CPDR instructions will loop back to step one if $A - (HL) = 0 \ \& \ BC = 0$
   If the value was found them HL will be set to the byte after or before it depending on the direction being used.

Timing

For the non-repeating instructions, they take 16(4,4,3,5) T-States to execute.

For the repeating instructions, they take either 21(4,4,3,5,5) T-States when they loop and 16(4,4,3,5) T-States when terminating.

Also note, that for these instructions the timing is for each iteration, not for the entire run. So if LDIR is run with BC=4 then the number of T-States for the entire operation would take 79(21+21+21+16) T-States.

Flags Affected

Flags	s	z	-	h	-	p/v	-	-

s Set if result is negative
z Set if A = (HL)
h Borrow from bit 4, otherwise reset
p/v Non-repeating: Set if BC-1 != 0, otherwise reset
Repeating: N/A as BC=0 after instruction completes

Opcode Matrix

	Increment	Decrement
Single Search	CPI 2 16 EDA1	CPD 2 16 EDA9
Repeat Search	CPIR 2 21 EDB1	CPDR 2 21 EDB9

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Memory

3.9 - Input/Output

Input Output instructions

3.9.1 - IN A, (n)

Read from port and store in A

$A \longleftarrow (n)$

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

This instruction places *n* onto the bottom half (A0...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...A15) of the address bus at this time. One byte from the selected port is placed on the data bus and written to the Accumulator (Register A).

Flags Affected

None.

Opcode Matrix

	A
IN (n)	IN A,(n) 2 11 DBnn

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Special

3.9.2 - IN r,(C)

Read from port in C and store in a specific register

7	6	5	4	3	2	1	0

$r \longleftarrow (C)$

IN r, (C)

1	1	1	0	1	1	0	1	ED
0	1	r			0	0	0	

$F \longleftarrow (C)$

IN F, (C)

1	1	1	0	1	1	0	1	ED
0	1	1	1	0	0	0	0	70

Registers

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

The contents of Register C are placed on the bottom half (A0...7) 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...A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to register r in the CPU.

> There is an undocumented code where r=%110 which sets the flag register.
>
> This is actually documented in Zilog's Z80 CPU User Manual, 2016 edition Page 296. For this reason it's included on this page and not in the Undocumented instruction section.

Flags Affected

Flags	s	z	-	h	-	p/v	n	-

- **s** set if input data is negative
- **z** set if input data is 0
- **h** reset
- **p/v** set if parity is even, reset if odd
- **n** reset

Opcode Matrix

	A	B	C	D	E	H	L	F
IN (C)	IN A,(C) 2 12 ED7B	IN B,(C) 2 12 ED40	IN C,(C) 2 12 ED48	IN D,(C) 2 12 ED50	IN E,(C) 2 12 ED58	IN H,(C) 2 12 ED60	IN L,(C) 2 12 ED68	IN F,(C) 2 12 ED70

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Special	Undocumented

3.9.3 - OUT (C), r

Write r to a port

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

$(C) \longleftarrow r$

OUT (C), r

Registers

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

$(C) \longleftarrow F$

OUT (C), F

1	1	1	0	1	1	0	1	ED
0	1	1	1	0	0	0	1	71

The contents of Register C are placed on the bottom half (A0...7) 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...A15) of the address bus at this time.

Then the byte contained in r is placed on the data bus and written to the selected peripheral device.

> There is an undocumented code where r=%110 which writes the flag register.
>
> Unlike it's IN F, (C) counterpart, this instruction is completely undocumented, but it's here not in the undocumented section to be consistent.

Flags Affected

None.

Opcode Matrix

	A	B	C	D	E	H	L	F
OUT (C)	OUT (C),A 2 12 ED79	OUT (C),B 2 12 ED41	OUT (C),C 2 12 ED49	OUT (C),D 2 12 ED51	OUT (C),E 2 12 ED59	OUT (C),H 2 12 ED61	OUT (C),L 2 12 ED69	OUT (C),F 2 12 ED71

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Special	Undocumented

3.9.4 - OUT (n), A

Write A to a port

$(n) \longleftarrow A$

7	6	5	4	3	2	1	0	
1	1	0	1	0	0	1	1	D3
n								

This instruction places *n* onto the bottom half (A0...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...A15) of the address bus at this time.

Then the byte contained in the Accumulator is placed on the data bus and written to the selected peripheral device.

Flags Affected

None.

Opcode Matrix

	A
OUT (n)	OUT (n),A 2 11 D3nn

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Special

3.9.5 - Block read from port

$$\left.\begin{array}{l}(HL) \longleftarrow (C)\\ HL \longleftarrow HL + 1 \text{ if } D = 0\\ HL \longleftarrow HL - 1 \text{ if } D = 1\\ B \longleftarrow B - 1\end{array}\right\} \text{repeat while } L = 1 \,\&\, B = 0$$

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	0	1	ED
1	0	1	L	D	0	1	0	

D 0=Increment, 1=Decrement HL after each iteration

L If set then if $B = 0$ then $PC \longleftarrow PC - 2$ so that the instruction is repeated.

The contents of Register C are placed on the bottom half (A0...A7) of the address bus to select the I/O device at one of 256 possible ports.

Register B can be used as a byte counter, and its contents are placed on the top half (A8...15) 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 to the corresponding location of memory.

Finally, the byte counter is decremented and register pair HL is incremented.

Flags Affected

Flags	-	z	-	-	-	-	n	-

z set if B = 0, always true for repeat operations

n set

Opcode Matrix

	Increment	Decrement
Single	INI 2 16 EDA2	IND 2 16 EDAA
Repeat	INIR 2 21 EDB2	INDR 2 21 EDBA

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Special

3.9.6 - Block write to port

$$\left.\begin{array}{l}(C) \longleftarrow (HL)\\ HL \longleftarrow HL + 1 \text{ if } D = 0\\ HL \longleftarrow HL - 1 \text{ if } D = 1\\ B \longleftarrow B - 1\end{array}\right\} \text{repeat while } L = 1 \,\&\, B = 0$$

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	0	1	ED
1	0	1	L	D	0	1	1	

D 0=Increment, 1=Decrement HL after each iteration

L If set then if $B = 0$ then $PC \longleftarrow PC - 2$ so that the instruction is repeated.

The contents of Register C are placed on the bottom half (A0...A7) of the address bus to select the I/O device at one of 256 possible ports.

Register B can be used as a byte counter, and its contents are placed on the top half (A8...15) of the address bus at this time.

Then one byte from the address pointed to by HL is placed on the data bus and written to the port.

Finally, the byte counter is decremented and register pair HL is incremented.

Flags Affected

Flags	-	z	-	-	-	-	n	-

z set if B = 0, always true for repeat operations

n set

Opcode Matrix

	Increment	Decrement
Single	OUTI 2 16 EDA3	OUTD 2 16 EDAB
Repeat	OUTIR 2 21 EDB3	OUTDR 2 21 EDBB

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Special

3.10 - Miscellaneous Instructions

Miscellaneous instructions

3.10.1 - NOP No Operation

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	00

Opcode Matrix

	NOP
OP	NOP 1 4 00

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Special

3.10.2 - CPL Invert Accumulator

$A \longleftarrow \overline{A}$

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

Flags Affected

Flags	-	-	-	h	-	-	n	-

h set

n set

Opcode Matrix

	CPL
OP	CPL 1 4 2F

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register

3.10.3 - NEG Negate Accumulator (two's compliment)

$A \longleftarrow 0 - A$

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	0	1	ED
0	1	0	0	0	1	0	0	44

Flags Affected

Flags	s	z	-	h	-	p/v	n	c

s set if result is negative

z set if result is 0

h set if borrow from bit 4

p/v set if Accumulator was 0x80 before operation

n set

c set if Accumulator was not 0x00 before operation

Opcode Matrix

	NEG
OP	NEG 2 4 ED44

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register

3.10.4 - HALT the cpu

7	6	5	4	3	2	1	0	
0	1	1	1	0	1	1	0	76

Opcode Matrix

	HALT
OP	HALT 1 4 76

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Special

3.10.5 - CCF Compliment Carry Flag

Invert Carry Flag

$CY \longleftarrow \overline{CY}$

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

Flags Affected

Flags	-	-	-	h	-	-	-	c

h previous carry is copied

c set if C was 0, reset if C was 1

Opcode Matrix

	CCF
OP	CCF 1 4 3F

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register

3.10.6 - SCF Set Carry Flag

Set Carry Flag

$CY \longleftarrow 1$

7	6	5	4	3	2	1	0	
0	0	1	1	0	1	1	1	37

Flags Affected

Flags	-	-	-	h	-	-	n	c

h reset

n reset

c set

Opcode Matrix

	CPL	NEG	CCF	SCF
OP	CPL 1 4 2F	NEG 2 4 ED44	CCF 1 4 3F	SCF 1 4 37

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register

3.10.7 - DI EI Interrupt enable

Enable/Disable interrupts

7	6	5	4	3	2	1	0

IFF ⟵ 0

DI

1	1	1	1	0	0	1	1

IFF ⟵ 1

EI

1	1	1	1	1	0	1	1

Opcode Matrix

	EI	DI
OP	EI 1 4 FB	DI 1 4 F3

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Interrupt

3.10.8 - IM Interrupt Mode

Select interrupt mode

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

Note: Only modes 0, 1 and 2 are valid for `IM n`.

Opcode Matrix

	IM0	IM1	IM2
OP	IM0 2 8 ED46	IM1 2 8 ED56	IM2 2 8 ED5E

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Interrupt

3.10.9 - DAA

Adjust accumulator for BCD addition and subtraction operations

@

7	6	5	4	3	2	1	0	
0	0	1	0	0	1	1	1	27

Flags Affected

Flags	-	z	-	h	-	p/v	-	c

z Set if Accumulator is 0

h Varies

p/v Set if Accumulator parity is even, reset if odd

c Varies

Opcode Matrix

	DAA
OP	DAA 1 4 27

Opcode Matrix Legend

Instruction Size bytes Cycle count Opcode hex	Register

3.11 - Undocumented Instructions

Undocumented instructions - use with care

Like most early microprocessors, the Z80 has it's own set of undocumented instructions.

Most of these either do something that's not useful, or they do something that would at first seem to be odd in why they were implemented in the first place.

In most instances, they exist due to how the processor is implemented in silicon. Where an instruction is decoded, there are free bits so if something tried to use that code then the processor would just do as it's told as it wouldn't know otherwise.

> Be aware, these usually work on a physical chip due to it requiring the actual instruction decoding to provide these instructions.
>
> They will most likely not work in an emulator as they would perform the decoding in software using lookup tables, so wouldn't implement anything that's not documented.

These may or may not work on actual chips. For example, on the 6502 there were plenty of undocumented instructions that were replaced in the 65C02 with NOP instructions.

These are provided here for reference only.

Overview

Most of the undocumented instructions fall under some simple rules:

CB

Only codes 0xCB30...0xCB37 are undocumented but implement a Shift Logical Left instruction where bit 0 is set post shift.

DDCB & FDCB

For opcodes with the 0xDDCB and 0xFDCB prefixes the instructions store the result in one of the 8-bit registers based on the lower 3 bits of the opcode: B=000, C=001, D=010, E=011, H=100, L=101 and A=111.

The officially documented codes all have 110 as the lower 3 bits and do not store the result in any register.

All of these instructions with the 0xDDCB prefix operate against the IX register (IY for 0xFDBC).

The only exception to this rule is opcodes 0x40...0x7F which are the bit text operations. As these only test the memory location they do not create a result so all the undocumented versions are identical to the official instructions.

DD & FD

Officially the 0xDD and 0xFD prefixes cause any instruction that references `(HL)` to instead work against the IX & IY registers with a displacement, 0xDD for IX and 0xFD for IY.

The undocumented instructions allows for instructions that refer to just H or L can also be used to access the upper or lower 8-bit components of IX and IY themselves.

ED

There are a few undocumented instructions with this prefix, but they simply emulate existing instructions.

The exception to this are the `IN F, (C)` and `OUT (C), F` instructions which are described below.

When is undocumented actually documented?

One oddity is the undocumented `IN F,(C)`$_{0xED70}$ instruction which performs an IN from an I/O port but stores the result into the Flags register. This instruction is actually documented in Zilogs own documentation (2016 PDF). For this reason, that instruction is listed on the IN r,(C) page and not in this section.

It's `OUT (C), F`$_{0xED71}$ equivalent is listed under OUT (C),r for consistency, even though that instruction is completely undocumented.

3.11.1 - Dual Shift Operations

Undocumented instructions that perform two actions at the same time

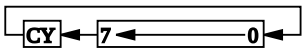
There are a few undocumented instructions that performs an action and then copies the result into a register.

For example the official `RLC (IX+nn)`$_{0xDDCBnn06}$ instruction operates on a specific memory address, however the undocumented `RLC B,(IX+nn)`$_{0xDDCBnn00}$ instruction does the same thing but then copies the result into the B register.

3.11.1.1 - RL Rotate bits left with Carry and store in register

Undocumented Rotate bits left with carry and store in register

This instruction performs an `RL (IX+dd)` or `RL (IX+dd)` operation but then also stores the result in a register as well as in the memory location.



7	6	5	4	3	2	1	0

RL r,(IX+d)

1	1	0	1	1	1	0	1	DD
1	1	0	0	1	0	1	1	CB
d								
0	0	0	1	0	r			

RL r,(IY+d)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
0	0	0	1	0	r			

Registers

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

Note: r=%110 does exist. It doesn't do a copy into a register as it's the existing official, documented instruction.

Opcode Matrix

	A	B	C	D	E	H	L
(IX+d)	RL A,(IX+d) DDCBnn17	RL B,(IX+d) DDCBnn10	RL C,(IX+d) DDCBnn11	RL D,(IX+d) DDCBnn12	RL E,(IX+d) DDCBnn13	RL H,(IX+d) DDCBnn14	RL L,(IX+d) DDCBnn15
(IY+d)	RL A,(IY+d) FDCBnn17	RL B,(IY+d) FDCBnn10	RL C,(IY+d) FDCBnn11	RL D,(IY+d) FDCBnn12	RL E,(IY+d) FDCBnn13	RL H,(IY+d) FDCBnn14	RL L,(IY+d) FDCBnn15

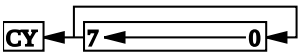
Opcode Matrix Legend

Instruction Opcode hex	Undocumented

3.11.1.2 - RLC Rotate bits left with Carry and store in register

Undocumented Rotate bits left with carry and store in register

This instruction performs an `RLC (IX+dd)` or `RLC (IX+dd)` operation but then also stores the result in a register as well as in the memory location.



7	6	5	4	3	2	1	0

RLC r,(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	0	r			

RLC r,(IY+d)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
0	0	0	0	0	r			

Registers

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

Note: r=%110 does exist. It doesn't do a copy into a register as it's the existing official, documented instruction.

Opcode Matrix

	A	B	C	D	E	H	L
(IX+d)	RLC A,(IX+d) DDCBnn07	RLC B,(IX+d) DDCBnn00	RLC C,(IX+d) DDCBnn01	RLC D,(IX+d) DDCBnn02	RLC E,(IX+d) DDCBnn03	RLC H,(IX+d) DDCBnn04	RLC L,(IX+d) DDCBnn05
(IY+d)	RLC A,(IY+d) FDCBnn07	RLC B,(IY+d) FDCBnn00	RLC C,(IY+d) FDCBnn01	RLC D,(IY+d) FDCBnn02	RLC E,(IY+d) FDCBnn03	RLC H,(IY+d) FDCBnn04	RLC L,(IY+d) FDCBnn05

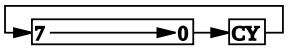
Opcode Matrix Legend

Instruction Opcode hex	Undocumented

3.11.1.3 - RR Rotate bits right with Carry and store in register

Undocumented Rotate bits right with carry and store in register

This instruction performs an `RR (IX+dd)` or `RR (IX+dd)` operation but then also stores the result in a register as well as in the memory location.



7	6	5	4	3	2	1	0

RR r,(IX+d)

1	1	0	1	1	1	0	1	DD
1	1	0	0	1	0	1	1	CB
d								
0	0	0	1	1	r			

RR r,(IY+d)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
0	0	0	1	1	r			

Registers

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

Note: r=%110 does exist. It doesn't do a copy into a register as it's the existing official, documented instruction.

Opcode Matrix

	A	B	C	D	E	H	L
(IX+d)	RR A,(IX+d) DDCBnn1F	RR B,(IX+d) DDCBnn18	RR C,(IX+d) DDCBnn19	RR D,(IX+d) DDCBnn1A	RR E,(IX+d) DDCBnn1B	RR H,(IX+d) DDCBnn1C	RR L,(IX+d) DDCBnn1D
(IY+d)	RR A,(IY+d) FDCBnn1F	RR B,(IY+d) FDCBnn18	RR C,(IY+d) FDCBnn19	RR D,(IY+d) FDCBnn1A	RR E,(IY+d) FDCBnn1B	RR H,(IY+d) FDCBnn1C	RR L,(IY+d) FDCBnn1D

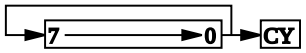
Opcode Matrix Legend

Instruction Opcode hex	Undocumented

3.11.1.4 - RRC Rotate bits right with Carry and store in register

Undocumented Rotate bits right with carry and store in register

This instruction performs an `RRC (IX+dd)` or `RRC (IX+dd)` operation but then also stores the result in a register as well as in the memory location.



7	6	5	4	3	2	1	0

RRC r,(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	r			

RRC r,(IY+d)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
0	0	0	0	1	r			

Registers

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

Note: r=%110 does exist. It doesn't do a copy into a register as it's the existing official, documented instruction.

Opcode Matrix

	A	B	C	D	E	H	L
(IX+d)	RRC A,(IX+d) DDCBnn0F	RRC B,(IX+d) DDCBnn08	RRC C,(IX+d) DDCBnn09	RRC D,(IX+d) DDCBnn0A	RRC E,(IX+d) DDCBnn0B	RRC H,(IX+d) DDCBnn0C	RRC L,(IX+d) DDCBnn0D
(IY+d)	RRC A,(IY+d) FDCBnn0F	RRC B,(IY+d) FDCBnn08	RRC C,(IY+d) FDCBnn09	RRC D,(IY+d) FDCBnn0A	RRC E,(IY+d) FDCBnn0B	RRC H,(IY+d) FDCBnn0C	RRC L,(IY+d) FDCBnn0D

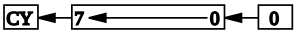
Opcode Matrix Legend

Instruction Opcode hex	Undocumented

3.11.1.5 - SLA Shift bits left with Carry and store in register

Undocumented Shift bits left with carry and store in register

This instruction performs an `SLA (IX+dd)` or `SLA (IX+dd)` operation but then also stores the result in a register as well as in the memory location.



7	6	5	4	3	2	1	0

SLA r,(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	r			

SLA r,(IY+d)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
0	0	1	0	0	r			

Registers

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

Note: r=%110 does exist. It doesn't do a copy into a register as it's the existing official, documented instruction.

Opcode Matrix

	A	B	C	D	E	H	L
(IX+d)	SLA A,(IX+d) DDCBnn27	SLA B,(IX+d) DDCBnn20	SLA C,(IX+d) DDCBnn21	SLA D,(IX+d) DDCBnn22	SLA E,(IX+d) DDCBnn23	SLA H,(IX+d) DDCBnn24	SLA L,(IX+d) DDCBnn25
(IY+d)	SLA A,(IY+d) FDCBnn27	SLA B,(IY+d) FDCBnn20	SLA C,(IY+d) FDCBnn21	SLA D,(IY+d) FDCBnn22	SLA E,(IY+d) FDCBnn23	SLA H,(IY+d) FDCBnn24	SLA L,(IY+d) FDCBnn25

Opcode Matrix Legend

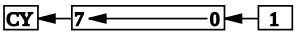
Instruction Opcode hex	Undocumented

3.11.1.6 - SLL Shift left Logical and store in register

Undocumented Shift left logical and store in register

This instruction performs an `SLL (IX+dd)` or `SLL (IX+dd)` operation but then also stores the result in a register as well as in the memory location.

Note: This is an undocumented extension to an undocumented instruction.



7	6	5	4	3	2	1	0

SLL r,(IX+d)

1	1	0	1	1	1	0	1	DD
1	1	0	0	1	0	1	1	CB
d								
0	0	1	1	0	r			

SLL r,(IY+d)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
0	0	1	1	0	r			

Registers

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

Note: r=%110 does exist. It doesn't do a copy into a register as it's the existing official, documented instruction.

Opcode Matrix

	A	B	C	D	E	H	L
(IX+d)	SLL A,(IX+d) DDCBnn37	SLL B,(IX+d) DDCBnn30	SLL C,(IX+d) DDCBnn31	SLL D,(IX+d) DDCBnn32	SLL E,(IX+d) DDCBnn33	SLL H,(IX+d) DDCBnn34	SLL L,(IX+d) DDCBnn35
(IY+d)	SLL A,(IY+d) FDCBnn37	SLL B,(IY+d) FDCBnn30	SLL C,(IY+d) FDCBnn31	SLL D,(IY+d) FDCBnn32	SLL E,(IY+d) FDCBnn33	SLL H,(IY+d) FDCBnn34	SLL L,(IY+d) FDCBnn35

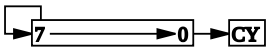
Opcode Matrix Legend

Instruction Opcode hex	Undocumented

3.11.1.7 - SRA Rotate bits right with Carry and store in register

Undocumented Rotate bits right with carry and store in register

This instruction performs an `SRA (IX+dd)` or `SRA (IX+dd)` operation but then also stores the result in a register as well as in the memory location.



7	6	5	4	3	2	1	0

SRA r,(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	r			

SRA r,(IY+d)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
0	0	1	0	1	r			

Registers

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

Note: r=%110 does exist. It doesn't do a copy into a register as it's the existing official, documented instruction.

Opcode Matrix

	A	B	C	D	E	H	L
(IX+d)	SRA A,(IX+d) DDCBnn2F	SRA B,(IX+d) DDCBnn28	SRA C,(IX+d) DDCBnn29	SRA D,(IX+d) DDCBnn2A	SRA E,(IX+d) DDCBnn2B	SRA H,(IX+d) DDCBnn2C	SRA L,(IX+d) DDCBnn2D
(IY+d)	SRA A,(IY+d) FDCBnn2F	SRA B,(IY+d) FDCBnn28	SRA C,(IY+d) FDCBnn29	SRA D,(IY+d) FDCBnn2A	SRA E,(IY+d) FDCBnn2B	SRA H,(IY+d) FDCBnn2C	SRA L,(IY+d) FDCBnn2D

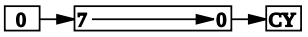
Opcode Matrix Legend

Instruction Opcode hex	Undocumented

3.11.1.8 - SRL Rotate bits right with Carry and store in register

Undocumented Rotate bits right with carry and store in register

This instruction performs an `SRL (IX+dd)` or `SRL (IX+dd)` operation but then also stores the result in a register as well as in the memory location.



7	6	5	4	3	2	1	0

SRL r,(IX+d)

1	1	0	1	1	1	0	1	DD
1	1	0	0	1	0	1	1	CB
d								
0	0	1	1	1	r			

SRL r,(IY+d)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
0	0	1	1	1	r			

Registers

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

Note: r=%110 does exist. It doesn't do a copy into a register as it's the existing official, documented instruction.

Opcode Matrix

	A	B	C	D	E	H	L
(IX+d)	SRL A,(IX+d) DDCBnn3F	SRL B,(IX+d) DDCBnn38	SRL C,(IX+d) DDCBnn39	SRL D,(IX+d) DDCBnn3A	SRL E,(IX+d) DDCBnn3B	SRL H,(IX+d) DDCBnn3C	SRL L,(IX+d) DDCBnn3D
(IY+d)	SRL A,(IY+d) FDCBnn3F	SRL B,(IY+d) FDCBnn38	SRL C,(IY+d) FDCBnn39	SRL D,(IY+d) FDCBnn3A	SRL E,(IY+d) FDCBnn3B	SRL H,(IY+d) FDCBnn3C	SRL L,(IY+d) FDCBnn3D

Opcode Matrix Legend

Instruction Opcode hex	Undocumented

3.11.2 - IX and IY registers

Undocumented instructions for IX and IY registers

If an opcode works with the Registers HL, H or L then if that opcode is prefixed by 0xDD then it will also work on the appropriate IX, IX_h or IX_l registers, with some exceptions.

The 0xFD prefix would also work but for the IY, IY_h or IY_l registers

The exceptions are instructions like `LD H,IX`$_h$ or `LD L,IY`$_h$ where it isn't clear from the opcode which register the 0xFD or 0xDD prefix should operate against.

3.11.2.1 - LD IX undocumented instructions

Undocumented instructions for LD IX

Opcode Matrix

	A	B	C	D	E	n	IXh	IXl
A							LD A,IXh DD7C	LD A,IXl DD7D
B							LD B,IXh DD44	LD B,IXl DD45
C							LD C,IXh DD4C	LD C,IXl DD4D
D							LD D,IXh DD54	LD D,IXl DD55
E							LD E,IXh DD5C	LD E,IXl DD5D
IXh	LD IXh,A DD67	LD IXh,B DD60	LD IXh,C DD61	LD IXh,D DD62	LD IXh,E DD63	LD IXh,n DD26nn	LD IXh,IHh DD64	LD IXh,IHl DD65
IXl	LD IXl,A DD6F	LD IXl,B DD68	LD IXl,C DD69	LD IXl,D DD6A	LD IXl,E DD6B	LD IXl,n DD2Enn	LD IXl,IHh DD6C	LD IXl,IHl DD6D

Opcode Matrix Legend

Instruction Opcode hex	Undocumented

3.11.2.2 - LD IY undocumented instructions

Undocumented instructions for LD IY

Opcode Matrix

	A	B	C	D	E	n	IYh	IYl
A							LD A,IYh FD7C	LD A,IYl FD7D
B							LD B,IYh FD44	LD B,IYl FD45
C							LD C,IYh FD4C	LD C,IYl FD4D
D							LD D,IYh FD54	LD D,IYl FD55
E							LD E,IYh FD5C	LD E,IYl FD5D
IYh	LD IYh,A FD67	LD IYh,B FD60	LD IYh,C FD61	LD IYh,D FD62	LD IYh,E FD63	LD IYh,n FD26nn	LD IYh,IHh FD64	LD IYh,IHl FD65
IYl	LD IYl,A FD6F	LD IYl,B FD68	LD IYl,C FD69	LD IYl,D FD6A	LD IYl,E FD6B	LD IYl,n FD2Enn	LD IYl,IHh FD6C	LD IYl,IHl FD6D

Opcode Matrix Legend

Instruction Opcode hex	Undocumented

3.11.2.3 - Undocumented Math instructions with the IX register

Undocumented math instructions for IX register

Opcode Matrix

	INC	DEC	ADD A	ADC A	SUB	SBC A	AND	XOR	OR	CP
IXh	INC IXh DD24	DEC IXh DD25	ADD A,IXh DD84	ADC A,IXh DD8C	SUB IXh DD94	SBC A,IXh DD9C	AND IXh DDA4	XOR IXh DDAC	OR IXh DDB4	CP IXh DDBC
IXl	INC IXl DD2C	DEC IXl DD2D	ADD A,IXl DD85	ADC A,IXl DD8D	SUB IXl DD95	SBC A,IXl DD9D	AND IXl DDA5	XOR IXl DDAD	OR IXl DDB5	CP IXl DDBD

Opcode Matrix Legend

Instruction Opcode hex	Undocumented

3.11.2.4 - Undocumented Math instructions with the IY register

Undocumented math instructions for IY register

Opcode Matrix

	INC	DEC	ADD A	ADC A	SUB	SBC A	AND	XOR	OR	CP
IYh	INC IYh FD24	DEC IYh FD25	ADD A,IYh FD84	ADC A,IYh FD8C	SUB IYh FD94	SBC A,IYh FD9C	AND IYh FDA4	XOR IYh FDAC	OR IYh FDB4	CP IYh FDBC
IYl	INC IYl FD2C	DEC IYl FD2D	ADD A,IYl FD85	ADC A,IYl FD8D	SUB IYl FD95	SBC A,IYl FD9D	AND IYl FDA5	XOR IYl FDAD	OR IYl FDB5	CP IYl FDBD

Opcode Matrix Legend

Instruction Opcode hex	Undocumented

3.11.3 - SLL Shift Left Logical

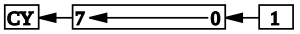
Undocumented instruction to perform a logical left shift

The block CB30...CB37 is missing from the official list.

These instructions, usually denoted by the mnemonic SLL, Shift Left Logical, shift left the operand and make bit 0 always one.

These instructions are quite commonly used. For example, Bounder and Enduro Racer use them.

Some documents list this as `SL1` instead of `SLL` due to it setting bit 0.



7	6	5	4	3	2	1	0

SLL r

1	1	0	0	1	0	1	1	CB
0	0	0	0	0	r			

SLL (HL)

1	1	0	0	1	0	1	1	CB
0	0	0	0	0	1	1	0	06

SLL (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	0	1	1	0	06

SLL (IY+d)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
0	0	0	0	0	1	1	0	06

Registers

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

Opcode Matrix

	A	B	C	D	E	H	L	(HL)	(IX+dd)	(IY+dd)
SLL	SLL A CB37	SLL B CB30	SLL C CB31	SLL D CB32	SLL E CB33	SLL H CB34	SLL L CB35	SLL (HL) CB36	SLL (IX+dd) DDCBnn36	SLL (IY+dd) FDCBnn36

Opcode Matrix Legend

Instruction Opcode hex	Undocumented

3.11.4 - Test bit in (IX+d)

Undocumented BIT n,(IX+d)

Similar to the RES and SET instructions, there are undocumented instructions for BIT. Unlike the other, as BIT only tests a bit and does not change anything, these opcodes have the same behaviour to the officially documented BIT instruction.

7	6	5	4	3	2	1	0

$Z \longleftarrow \overline{(IX+d)_b}$

BIT b, (IX+d)

1	1	0	1	1	1	0	1	DD
1	1	0	0	1	0	1	1	CB
d								
0	1	b			r			

$Z \longleftarrow \overline{(IY+d)_b}$

BIT b, (IY+d)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
0	1	b			r			

Registers

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

Bits

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

Z is set if the specified bit in the source is 0, otherwise it is cleared.

r=%110 does exist, it is the official, documented operation.

3.11.5 - RES Reset bit in (IX+d) and copy into register r

Undocumented Reset bit in (IX+d) and copy into register r

There are a few undocumented instructions that performs an action and then copies the result into a register.

For example the official `RES 0,(IX+nn)` instruction resets bit 0 on a specific memory address, however the undocumented `RES B,0,(IX+nn)`$_{0xDDCBnn00}$ instruction does the same thing but then copies the result into the B register.

$(IX + d)_b \longleftarrow 0$
$r \longleftarrow (IX + d)$

7	6	5	4	3	2	1	0	
1	1	0	1	1	1	0	1	DD
1	1	0	0	1	0	1	1	CB
d								
1	0	b			r			

Registers

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

Bits

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

Z is set if the specified bit in the source is 0, otherwise it is cleared.

The result is stored both in memory and the specified register.

r=%110 does exist, it is the official documented operation with no auto-copy to a register.

Opcode Matrix

	BIT 0	BIT 1	BIT 2	BIT 3	BIT 4	BIT 5	BIT 6	BIT 7
A	RES A,0,(IX+nn) DDCBnn87	RES A,1,(IX+nn) DDCBnn8F	RES A,2,(IX+nn) DDCBnn97	RES A,3,(IX+nn) DDCBnn9F	RES A,4,(IX+nn) DDCBnnA7	RES A,5,(IX+nn) DDCBnnAF	RES A,6,(IX+nn) DDCBnnB7	RES A,7,(IX+nn) DDCBnnBF
B	RES B,0,(IX+nn) DDCBnn80	RES B,1,(IX+nn) DDCBnn88	RES B,2,(IX+nn) DDCBnn90	RES B,3,(IX+nn) DDCBnn98	RES B,4,(IX+nn) DDCBnnA0	RES B,5,(IX+nn) DDCBnnA8	RES B,6,(IX+nn) DDCBnnB0	RES B,7,(IX+nn) DDCBnnB8
C	RES C,0,(IX+nn) DDCBnn81	RES C,1,(IX+nn) DDCBnn89	RES C,2,(IX+nn) DDCBnn91	RES C,3,(IX+nn) DDCBnn99	RES C,4,(IX+nn) DDCBnnA1	RES C,5,(IX+nn) DDCBnnA9	RES C,6,(IX+nn) DDCBnnB1	RES C,7,(IX+nn) DDCBnnB9
D	RES D,0,(IX+nn) DDCBnn82	RES D,1,(IX+nn) DDCBnn8A	RES D,2,(IX+nn) DDCBnn92	RES D,3,(IX+nn) DDCBnn9A	RES D,4,(IX+nn) DDCBnnA2	RES D,5,(IX+nn) DDCBnnAA	RES D,6,(IX+nn) DDCBnnB2	RES D,7,(IX+nn) DDCBnnBA
E	RES E,0,(IX+nn) DDCBnn83	RES E,1,(IX+nn) DDCBnn8B	RES E,2,(IX+nn) DDCBnn93	RES E,3,(IX+nn) DDCBnn9B	RES E,4,(IX+nn) DDCBnnA3	RES E,5,(IX+nn) DDCBnnAB	RES E,6,(IX+nn) DDCBnnB3	RES E,7,(IX+nn) DDCBnnBB
H	RES H,0,(IX+nn) DDCBnn84	RES H,1,(IX+nn) DDCBnn8C	RES H,2,(IX+nn) DDCBnn94	RES H,3,(IX+nn) DDCBnn9C	RES H,4,(IX+nn) DDCBnnA4	RES H,5,(IX+nn) DDCBnnAC	RES H,6,(IX+nn) DDCBnnB4	RES H,7,(IX+nn) DDCBnnBC
L	RES L,0,(IX+nn) DDCBnn85	RES L,1,(IX+nn) DDCBnn8D	RES L,2,(IX+nn) DDCBnn95	RES L,3,(IX+nn) DDCBnn9D	RES L,4,(IX+nn) DDCBnnA5	RES L,5,(IX+nn) DDCBnnAD	RES L,6,(IX+nn) DDCBnnB5	RES L,7,(IX+nn) DDCBnnBD

Opcode Matrix Legend

Instruction Opcode hex	Undocumented

3.11.6 - RES Reset bit in (IY+d) and copy into register r

Undocumented Reset bit in (IY+d) and copy into register r

There are a few undocumented instructions that performs an action and then copies the result into a register.

For example the official `RES 0,(IY+nn)` instruction resets bit 0 on a specific memory address, however the undocumented `RES B,0,(IY+nn)`$_{0xFDCBnn00}$ instruction does the same thing but then copies the result into the B register.

$(IY + d)_b \longleftarrow 0$
$r \longleftarrow (IY + d)$

7	6	5	4	3	2	1	0	
1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
1	0	b			r			

Registers

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

Bits

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

Z is set if the specified bit in the source is 0, otherwise it is cleared.

The result is stored both in memory and the specified register.

r=%110 does exist, it is the official documented operation with no auto-copy to a register.

Opcode Matrix

	BIT 0	BIT 1	BIT 2	BIT 3	BIT 4	BIT 5	BIT 6	BIT 7
A	RES A,0,(IY+nn) FDCBnn87	RES A,1,(IY+nn) FDCBnn8F	RES A,2,(IY+nn) FDCBnn97	RES A,3,(IY+nn) FDCBnn9F	RES A,4,(IY+nn) FDCBnnA7	RES A,5,(IY+nn) FDCBnnAF	RES A,6,(IY+nn) FDCBnnB7	RES A,7,(IY+nn) FDCBnnBF
B	RES B,0,(IY+nn) FDCBnn80	RES B,1,(IY+nn) FDCBnn88	RES B,2,(IY+nn) FDCBnn90	RES B,3,(IY+nn) FDCBnn98	RES B,4,(IY+nn) FDCBnnA0	RES B,5,(IY+nn) FDCBnnA8	RES B,6,(IY+nn) FDCBnnB0	RES B,7,(IY+nn) FDCBnnB8
C	RES C,0,(IY+nn) FDCBnn81	RES C,1,(IY+nn) FDCBnn89	RES C,2,(IY+nn) FDCBnn91	RES C,3,(IY+nn) FDCBnn99	RES C,4,(IY+nn) FDCBnnA1	RES C,5,(IY+nn) FDCBnnA9	RES C,6,(IY+nn) FDCBnnB1	RES C,7,(IY+nn) FDCBnnB9
D	RES D,0,(IY+nn) FDCBnn82	RES D,1,(IY+nn) FDCBnn8A	RES D,2,(IY+nn) FDCBnn92	RES D,3,(IY+nn) FDCBnn9A	RES D,4,(IY+nn) FDCBnnA2	RES D,5,(IY+nn) FDCBnnAA	RES D,6,(IY+nn) FDCBnnB2	RES D,7,(IY+nn) FDCBnnBA
E	RES E,0,(IY+nn) FDCBnn83	RES E,1,(IY+nn) FDCBnn8B	RES E,2,(IY+nn) FDCBnn93	RES E,3,(IY+nn) FDCBnn9B	RES E,4,(IY+nn) FDCBnnA3	RES E,5,(IY+nn) FDCBnnAB	RES E,6,(IY+nn) FDCBnnB3	RES E,7,(IY+nn) FDCBnnBB
H	RES H,0,(IY+nn) FDCBnn84	RES H,1,(IY+nn) FDCBnn8C	RES H,2,(IY+nn) FDCBnn94	RES H,3,(IY+nn) FDCBnn9C	RES H,4,(IY+nn) FDCBnnA4	RES H,5,(IY+nn) FDCBnnAC	RES H,6,(IY+nn) FDCBnnB4	RES H,7,(IY+nn) FDCBnnBC
L	RES L,0,(IY+nn) FDCBnn85	RES L,1,(IY+nn) FDCBnn8D	RES L,2,(IY+nn) FDCBnn95	RES L,3,(IY+nn) FDCBnn9D	RES L,4,(IY+nn) FDCBnnA5	RES L,5,(IY+nn) FDCBnnAD	RES L,6,(IY+nn) FDCBnnB5	RES L,7,(IY+nn) FDCBnnBD

Opcode Matrix Legend

Instruction Opcode hex	Undocumented

3.11.7 - SET bit in (IX+d) and copy into register r

Undocumented SET bit in (IX+d) and copy into register r

There are a few undocumented instructions that performs an action and then copies the result into a register.

For example the official `SET 0,(IX+nn)` instruction sets bit 0 on a specific memory address, however the undocumented `SET B,0,(IX+nn)`$_{0xDDCBnn00}$ instruction does the same thing but then copies the result into the B register.

$(IX + d)_b \longleftarrow 0$
$r \longleftarrow (IX + d)$

7	6	5	4	3	2	1	0	
1	1	1	1	1	1	0	1	DD
1	1	0	0	1	0	1	1	CB
d								
1	1	b			r			

Registers

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

Bits

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

Z is set if the specified bit in the source is 0, otherwise it is cleared.

The result is stored both in memory and the specified register.

r=%110 does exist, it is the official documented operation with no auto-copy to a register.

Opcode Matrix

	BIT 0	BIT 1	BIT 2	BIT 3	BIT 4	BIT 5	BIT 6	BIT 7
A	SET A,0,(IX+nn) DDCBnnC7	SET A,1,(IX+nn) DDCBnnCF	SET A,2,(IX+nn) DDCBnnD7	SET A,3,(IX+nn) DDCBnnDF	SET A,4,(IX+nn) DDCBnnE7	SET A,5,(IX+nn) DDCBnnEF	SET A,6,(IX+nn) DDCBnnF7	SET A,7,(IX+nn) DDCBnnFF
B	SET B,0,(IX+nn) DDCBnnC0	SET B,1,(IX+nn) DDCBnnC8	SET B,2,(IX+nn) DDCBnnD0	SET B,3,(IX+nn) DDCBnnD8	SET B,4,(IX+nn) DDCBnnE0	SET B,5,(IX+nn) DDCBnnE8	SET B,6,(IX+nn) DDCBnnF0	SET B,7,(IX+nn) DDCBnnF8
C	SET C,0,(IX+nn) DDCBnnC1	SET C,1,(IX+nn) DDCBnnC9	SET C,2,(IX+nn) DDCBnnD1	SET C,3,(IX+nn) DDCBnnD9	SET C,4,(IX+nn) DDCBnnE1	SET C,5,(IX+nn) DDCBnnE9	SET C,6,(IX+nn) DDCBnnF1	SET C,7,(IX+nn) DDCBnnF9
D	SET D,0,(IX+nn) DDCBnnC2	SET D,1,(IX+nn) DDCBnnCA	SET D,2,(IX+nn) DDCBnnD2	SET D,3,(IX+nn) DDCBnnDA	SET D,4,(IX+nn) DDCBnnE2	SET D,5,(IX+nn) DDCBnnEA	SET D,6,(IX+nn) DDCBnnF2	SET D,7,(IX+nn) DDCBnnFA
E	SET E,0,(IX+nn) DDCBnnC3	SET E,1,(IX+nn) DDCBnnCB	SET E,2,(IX+nn) DDCBnnD3	SET E,3,(IX+nn) DDCBnnDB	SET E,4,(IX+nn) DDCBnnE3	SET E,5,(IX+nn) DDCBnnEB	SET E,6,(IX+nn) DDCBnnF3	SET E,7,(IX+nn) DDCBnnFB
H	SET H,0,(IX+nn) DDCBnnC4	SET H,1,(IX+nn) DDCBnnCC	SET H,2,(IX+nn) DDCBnnD4	SET H,3,(IX+nn) DDCBnnDC	SET H,4,(IX+nn) DDCBnnE4	SET H,5,(IX+nn) DDCBnnEC	SET H,6,(IX+nn) DDCBnnF4	SET H,7,(IX+nn) DDCBnnFC
L	SET L,0,(IX+nn) DDCBnnC5	SET L,1,(IX+nn) DDCBnnCD	SET L,2,(IX+nn) DDCBnnD5	SET L,3,(IX+nn) DDCBnnDD	SET L,4,(IX+nn) DDCBnnE5	SET L,5,(IX+nn) DDCBnnED	SET L,6,(IX+nn) DDCBnnF5	SET L,7,(IX+nn) DDCBnnFD

Opcode Matrix Legend

Instruction Opcode hex	Undocumented

3.11.8 - SET bit in (IY+d) and copy into register r

Undocumented SET bit in (IY+d) and copy into register r

There are a few undocumented instructions that performs an action and then copies the result into a register.

For example the official `SET 0,(IY+nn)` instruction sets bit 0 on a specific memory address, however the undocumented `SET B,0,(IY+nn)`$_{0xFDCBnn00}$ instruction does the same thing but then copies the result into the B register.

$(IY + d)_b \longleftarrow 0$
$r \longleftarrow (IY + d)$

7	6	5	4	3	2	1	0	
1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
d								
1	1	b			r			

Registers

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

Bits

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

Z is set if the specified bit in the source is 0, otherwise it is cleared.

The result is stored both in memory and the specified register.

r=%110 does exist, it is the official documented operation with no auto-copy to a register.

Opcode Matrix

	BIT 0	BIT 1	BIT 2	BIT 3	BIT 4	BIT 5	BIT 6	BIT 7
A	SET A,0,(IY+nn) FDCBnnC7	SET A,1,(IY+nn) FDCBnnCF	SET A,2,(IY+nn) FDCBnnD7	SET A,3,(IY+nn) FDCBnnDF	SET A,4,(IY+nn) FDCBnnE7	SET A,5,(IY+nn) FDCBnnEF	SET A,6,(IY+nn) FDCBnnF7	SET A,7,(IY+nn) FDCBnnFF
B	SET B,0,(IY+nn) FDCBnnC0	SET B,1,(IY+nn) FDCBnnC8	SET B,2,(IY+nn) FDCBnnD0	SET B,3,(IY+nn) FDCBnnD8	SET B,4,(IY+nn) FDCBnnE0	SET B,5,(IY+nn) FDCBnnE8	SET B,6,(IY+nn) FDCBnnF0	SET B,7,(IY+nn) FDCBnnF8
C	SET C,0,(IY+nn) FDCBnnC1	SET C,1,(IY+nn) FDCBnnC9	SET C,2,(IY+nn) FDCBnnD1	SET C,3,(IY+nn) FDCBnnD9	SET C,4,(IY+nn) FDCBnnE1	SET C,5,(IY+nn) FDCBnnE9	SET C,6,(IY+nn) FDCBnnF1	SET C,7,(IY+nn) FDCBnnF9
D	SET D,0,(IY+nn) FDCBnnC2	SET D,1,(IY+nn) FDCBnnCA	SET D,2,(IY+nn) FDCBnnD2	SET D,3,(IY+nn) FDCBnnDA	SET D,4,(IY+nn) FDCBnnE2	SET D,5,(IY+nn) FDCBnnEA	SET D,6,(IY+nn) FDCBnnF2	SET D,7,(IY+nn) FDCBnnFA
E	SET E,0,(IY+nn) FDCBnnC3	SET E,1,(IY+nn) FDCBnnCB	SET E,2,(IY+nn) FDCBnnD3	SET E,3,(IY+nn) FDCBnnDB	SET E,4,(IY+nn) FDCBnnE3	SET E,5,(IY+nn) FDCBnnEB	SET E,6,(IY+nn) FDCBnnF3	SET E,7,(IY+nn) FDCBnnFB
H	SET H,0,(IY+nn) FDCBnnC4	SET H,1,(IY+nn) FDCBnnCC	SET H,2,(IY+nn) FDCBnnD4	SET H,3,(IY+nn) FDCBnnDC	SET H,4,(IY+nn) FDCBnnE4	SET H,5,(IY+nn) FDCBnnEC	SET H,6,(IY+nn) FDCBnnF4	SET H,7,(IY+nn) FDCBnnFC
L	SET L,0,(IY+nn) FDCBnnC5	SET L,1,(IY+nn) FDCBnnCD	SET L,2,(IY+nn) FDCBnnD5	SET L,3,(IY+nn) FDCBnnDD	SET L,4,(IY+nn) FDCBnnE5	SET L,5,(IY+nn) FDCBnnED	SET L,6,(IY+nn) FDCBnnF5	SET L,7,(IY+nn) FDCBnnFD

Opcode Matrix Legend

Instruction Opcode hex	Undocumented

4 - Decoding Instructions

How to decode instructions from binary

This section lists how the instructions are laid out at the bit level.

Normally if you are manually disassembling code you just need to use the list by Opcodes, however this section will be useful if you are implementing a Z80 emulator as you can see how the instruction decoding works including how the undocumented instructions work due to how the bits are organised.

How to use these decoding tables

To decode an opcode, convert it to binary then run through it from left to right, e.g. start at Bit 7 and move towards Bit 0.

As you run through the bits, start on the table from the top-left and go down then right as you find each bit. Bits are ordered with 0 first, then 1 & finally x which indicates that bit can be either 0 or 1.

When you find a match then go with that. If more than one entry matches then go for the one higher in the table as that will have higher precedence.

Z80 Instruction Decode table

To decode an instruction:

- If the opcode is CB, DD, ED or FD then go to that prefix page and decode the next byte.
- Using the table below, locate the pattern that matches the opcode. It's usually best to start from the left (bit 7) and go right until you find the opcode.
  An x means a bit that can be either 0 or 1, however check adjacent rows first and always take an entry that has a 0 or 1 as higher precedence to the X.

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	Nop
0	0	0	0	1	0	0	0	EX
0	0	0	1	x	0	0	0	Flow
0	0	0	x	x	1	1	1	Rotate
0	0	1	x	x	1	1	1	Misc
0	0	x	x	0	0	0	1	LD Instructions
0	0	x	x	0	0	1	1	Arithmetic Instructions
0	0	x	x	0	1	0	0	Arithmetic Instructions
0	0	x	x	0	1	0	1	Arithmetic Instructions
0	0	x	x	0	1	1	0	LD Instructions
0	0	x	x	1	0	0	1	Arithmetic Instructions
0	0	x	x	x	0	0	0	Flow
0	0	x	x	x	0	1	0	LD Instructions
0	1	1	1	0	1	1	0	Halt
0	1	x	x	x	x	x	x	LD Instructions
1	0	x	x	x	x	x	x	Arithmetic Instructions
1	1	0	0	0	0	1	1	Flow
1	1	0	0	1	0	1	1	CB Prefix
1	1	0	0	1	1	0	1	Flow
1	1	0	1	1	0	0	1	EXX
1	1	0	1	x	0	1	1	I/O
1	1	1	0	1	1	0	1	ED Prefix
1	1	1	0	x	0	1	1	EX
1	1	1	1	1	0	0	1	LD Instructions
1	1	1	1	x	0	1	1	Interrupts
1	1	x	0	1	0	0	1	Flow
1	1	x	1	1	1	0	1	DD FD Prefix
1	1	x	x	0	x	0	1	Stack Instructions
1	1	x	x	x	0	0	0	Flow
1	1	x	x	x	0	1	0	Flow
1	1	x	x	x	1	0	0	Flow
1	1	x	x	x	1	1	0	Arithmetic Instructions
1	1	x	x	x	1	1	1	Flow

Notes:

1. Halt 0x76 is where the invalid LD (HL),(HL) instruction would have been.

4.1 - Arithmetic Instructions

How to decode arithmetic instructions from binary

Opcodes with bits 7...5 set to 100 are the arithmetic instructions `ADD`, `ADC`, `SUB` and `SBC`. As are those starting with 7...6 set to 11 but ending with bits 2...0 set to 110. These instructions take an additional numeric operand after the opcode and use that instead of a register as the source.

Those with 7...5 set to 101 are the logic instructions `AND`, `XOR`, `OR` and `CP`.

Opcode format

7	6	5	4	3	2	1	0

Arithmetic with register as source, e.g. ADD A

1	0	0	A	F	r		

Logic with register as source, e.g. OR A

1	0	1	A	F	r		

8 bit number as source, e.g. SUB 5

1	1	X	A	F	1	1	0
n							

Registers

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

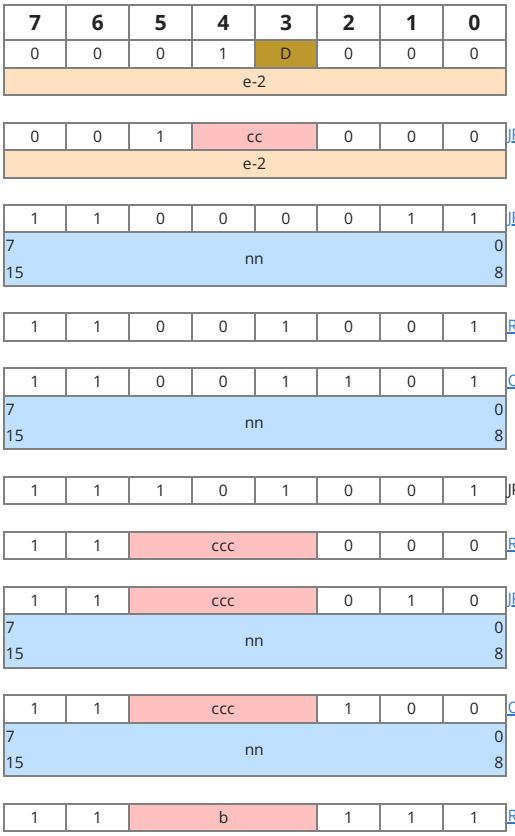
Lookup for A and F bits

7	6	5	A	F	Instruction	r
1	0	0	0	0	ADC r	register or 110 = (HL)
				1	ADD r	
			1	0	SBC r	
				1	SUB r	
		1	0	0	AND r	
				1	XOR r	
			1	0	OR r	
				1	CP r	
	1	0	0	0	ADC n	Always set to 110
				1	ADD n	
			1	0	SBC n	
				1	SUB n	
		1	0	0	AND n	
				1	XOR n	
			1	0	OR n	
				1	CP n	

4.2 - Program Flow Instructions

How to decode program flow instructions from binary

Opcode format



Conditions

cc	ccc	Abbrev	Condition	Flag
00	000	NZ	Non Zero	Z
01	001	Z	Zero	
10	010	NC	No Carry	C
11	011	C	Carry	
	100	PO	Parity Odd	P/V
	101	PE	Parity Even	
	110	P	Sign Positive	S
	111	M	Sign Negative	

Bits

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

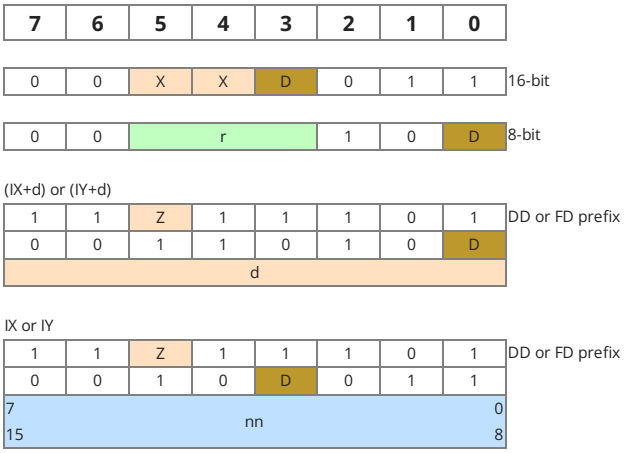
D

Instruction	D
DJNZ	0
JR	1

4.3 - Increment Decrement Instructions

How to decode increment and decrement instructions from binary

Opcode format



Registers

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

XX Register

Instruction	XX
BC	00
DE	01
HL	10
A	11

D direction

Direction	D
INC	0
DEC	1

Z

Register	Z
IX	0
IY	1

4.4 - LD Load instructions

How to decode ld instructions from binary

Opcode format

7	6	5	4	3	2	1	0

Set 1

0	0	0	B	0	1	0

Set 4

0	0	B	0	0	0	1

Set 2

0	0	1	B	0	1	0
7 15			nn			0 8

Set 3

0	0	b	1	1	0
n					

LD r, r'

0	1	r	r'

LD r, (HL)

0	1	r	1	1	0

LD SP,HL

1	1	1	1	1	0	0	1

Registers

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

Bits

Value	b	B
0	000	00
1	001	01
2	010	10
3	011	11
4	100	
5	101	
6	110	
7	111	

Set 1 store a in memory

4	3	Instruction
0	0	LD (BC),A
	1	LD (DE),A
1	0	LD A,(BC)
	1	LD A,(DE)

Set 2 store in memory

5	4	Instruction
0	0	LD (nn),HL
	1	LD (nn),A
1	0	LD HL,(nn)
	1	LD A,(nn)

Set 3 set to constant n

4	3	2	Instruction
0	0	0	LD B,n
		1	LD C,n
	1	0	LD D,n
		1	LD E,n
1	0	0	LD H,n
		1	LD L,n
	1	0	LD (HL),n
		1	LD A,n

Set 4 set to constant nnn

5	4	Instruction
0	0	LD BC,nn
	1	LD DE,nn
1	0	LD HL,nn
	1	LD SP,nn

4.5 - Miscelaneous Instructions

How to decode IO, EX and Interrupt instructions from binary

Only four rotate instructions are defined in the main opcode set, all the rest require the `CB` prefix.

Opcode format

7	6	5	4	3	2	1	0

IO

1	1	0	1	D	0	1	1

EXX

1	1	0	1	1	0	0	1

EX

1	1	1	0	W	0	1	1

Interrupts

1	1	1	1	E	0	1	1

D I/O Direction

Instruction	D
Out	0
In	1

W EX registers

Instruction	W
DE_HL	0
(SP)_HL	1

E Interrupt Enable

Instruction	E
DI	0
EI	1

4.6 - Rotate Instructions

How to decode rotate instructions from binary

Only four rotate instructions are defined in the main opcode set, all the rest require the `CB` prefix.

Opcode format

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

0	0	0	0	0	1	1	1	RLCA
0	0	0	1	0	1	1	1	RLA
0	0	0	0	0	1	1	1	RRCA
0	0	0	1	1	1	1	1	RRA

F Carry Flag

Instruction	F
With Carry	0
Without Carry	1

D Direction

Instruction	D
Left	0
Right	1

4.7 - Decoding CB Prefix

How the CB instruction prefix works

Instructions with the CB prefix consist of instructions that manipulate individual bits in a register or memory.

7	6	5	4	3	2	1	0	
0	0	0	0	0	x	x	x	RLC r
0	0	0	0	1	x	x	x	RRC r
0	0	0	1	0	x	x	x	RL
0	0	0	1	1	x	x	x	RR
0	0	1	0	0	x	x	x	SLA
0	0	1	0	1	x	x	x	SRA
0	0	1	1	0	x	x	x	SLL
0	0	1	1	1	x	x	x	SRL
0	1	x	x	x	x	x	x	BIT b,r
1	0	x	x	x	x	x	x	RES b,r
1	1	x	x	x	x	x	x	SET b,r

Operations with IX and IY registers

The operations here which operator on the `(HL)` register do also support use with the `IX` and `IY` registers with a relative offset. They are identical to the `(HL)` operation but with a DD or FD prefix.

Only instructions with the lower nibble set to 6 or E are documented. The other opcodes are undocumented.

Undocumented SLL instruction

Op codes CB30...CB37 are undocumented; but they do perform a shift left operation, placing a 1 in bit 0 and setting the carry flag to the original bit 7.

4.8 - Decoding ED Prefix

How the ED instruction prefix works

Instructions with the ED prefix consist of instructions that are not used as often as those in the main group.

7	6	5	4	3	2	1	0	
0	1	0	0	0	1	0	0	NEG
0	1	0	0	0	1	0	1	RETN
0	1	0	0	1	1	0	1	RETI
0	1	0	1	0	0	1	1	
0	1	0	1	1	1	1	0	IM2
0	1	0	x	0	1	1	0	IMx
0	1	0	x	0	1	1	1	LD
0	1	0	x	1	0	0	0	LD
0	1	1	0	0	1	1	1	RRD (HL)
0	1	1	0	1	0	0	0	LD
0	1	1	0	1	1	1	1	RLD (HL)
0	1	1	1	0	0	0	1	
0	1	1	1	1	0	0	1	OUT (C) A
0	1	x	x	0	0	1	0	SBC
0	1	x	x	1	0	1	0	ADC
0	1	x	x	x	0	0	0	IN r (C)
0	1	x	x	x	0	0	1	OUT (C) r
0	1	x	x	x	0	1	1	LD
1	0	1	x	x	0	0	x	Block Memory
1	0	1	x	x	0	1	x	Block IO

4.9 - Decoding DD and FD Prefixes

How the DD and FD instruction prefixes work

Instructions with either the `DD` or `FD` prefixes affect those instructions that operate against the memory addressed by `HL`, changing them to use either the `IX` or `IY` registers with an offset.

Instructions that refer directly to the `HL` register will then act directly against either `IX` or `IY`. For those that refer to `(HL)`, i.e. the memory pointed to by `HL` then the instructions use an additional relative offset that's added to either the `IX` or `IY` registers, and are written as `(IX+d)` or `(IY+d)`.

Instructions with the `DD` prefix use the `IX` register, whilst the `FD` prefix uses the `IY` register.

DDCB and FDCB Prefixes

The `DD` and `FD` prefixes extends though the `CB` prefix as it does for normal instructions. Just like the CB prefix

The format of the instruction also changes slightly as they change the behaviour of the existing instructions with the `CB` prefix. These instructions are all four bytes long with the third byte consisting of the offset.

For example: The `RLC (HL)` is encoded as CB06.

With the `DD` prefix this becomes `RLC (IX+d)` but the instruction is formatted as DDCBdd06. With the `FD` prefix this becomes `RLC (IY+d)`, formatted as FDCBdd06.

Note that the offset `d` is before the final part of the operand, not after as you might expect.

Decoder

All of these have either DD or FD as the previous prefix byte and a displacement immediately after them.

7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	1	LD
0	0	1	0	X	0	1	1	INC DEC
0	0	1	0	x	x	1	0	LD
0	0	1	1	0	1	0	D	IncDec
0	0	1	1	0	1	1	0	LD
0	0	x	x	1	0	0	1	ADD
0	1	0	0	0	1	0	x	INC DEC
0	1	1	1	0	1	1	0	
0	1	1	1	0	x	x	x	LD
0	1	x	x	x	1	1	0	LD
1	0	x	x	0	1	1	0	LD
1	1	0	0	1	0	1	1	CB Prefix
1	1	1	0	0	0	1	1	EX
1	1	1	0	0	x	0	1	Stack Instructions
1	1	1	0	1	0	0	1	Flow
1	1	1	1	1	0	0	1	LD

5 - Optimizing code

Things to try to optimise Z80 code

Writing code on an 8-bit microcomputer requires a skill that has been lost in the modern programming era. These days, developers are used to having Gigabytes of memory and processors that run at multiple Gigahertz.

In the microcomputer era we had far, far less resources. Processor clocks ran at 2 or 4 MHz, one thousandth of the clock speed of modern processors. If we were lucky we had 32K, 48K or 64K of memory to play with and that was it.

Because of this, we had to learn pretty quickly how to optimise our code to fit into memory. If we were lucky we could use a floppy disk to page in parts of the program as needed, but even then when a Cassette tape was the primary medium for a platform that wasn't even possible.

Optimisations at the machine code level would be a balance of reducing the size of code and having code run as fast as possible.

Sometimes you might sacrifice some memory for speed if the routine is important, for example it's doing a transform for some graphics in real time, but most of the time it's to reduce the memory used.

5.1 - Accumulator

Optimising use of the A register

Setting the Accumulator to 0

When dealing with loading 0 into the Accumulator, there's several ways to do it.

```
3E00            LD      A,0             ; Traditional way to set A to 0
AF              XOR     A               ; Anything xor itself is 0
97              SUB     A               ; A-A=0
```

The downside to the above options is that they also affect the flags. However, they are only 1 byte long not 2 and are both 3 T-states faster.

Inverting A

If inverting A, i.e. swapping each bit from 1 to 0 and vis-versa then instead of `XOR 0xFF` use `CPL` instead. It's both faster, 1 byte and that's all that instruction does.

```
EEFF            XOR     0xFF            ; A=A XOR 0xff
2F              CPL                     ; This instruction does exactly the same thing and nothing else!
```

The downside is that `CPL` does not affect the flags whilst `XOR` does.

5.2 - Comparisons

Optimising comparing numbers

A = 0

A common test is to see if A is 0, so you would expect to use `CP 0` to test for it.

Just like setting A to 0 we can compare quicker. In this case, instead of `CP 0` we can just use either `OR A` or `AND A` instead. Just 1 byte and 3 T-states faster.

```
FE00            CP      0               ; A-0 will set Z if A is also 0
A7              AND     A               ; Anything AND itself is itself but Z is set if A is 0
B7              OR      A               ; Anything OR itself is itself but Z is set if A is 0
```

For example, take this simple routine which writes a NULL terminated string pointed to by HL to the screen of the Sinclair ZX Spectrum:

```
                                        ; Print null terminated string at HL to the screen
.printStr       LD      A,(HL)          ; get next byte
                CP      0               ; check for null
                RET     Z               ; Stop when we get a null
                RST     2               ; print the character
                INC     HL              ; move to next character
                JR      printStr        ; loop back
```

The optimisation here is to replace `CP 0` with `OR A`

```
                                        ; Print null terminated string at HL to the screen
.printStr       LD      A,(HL)          ; get next byte
                OR      A               ; check for null
                RET     Z               ; Stop when we get a null
                RST     2               ; print the character
                INC     HL              ; move to next character
                JR      printStr        ; loop back
```

A = 1

Comparing A to 1 can also be done using `DEC A` instead of `CP 1`. By decrementing A, the Z flag will be set if A is now 0. Like above its faster and 1 byte, but it also alters A, so it's not really of any use unless you don't care about the value of A after the test.

```
FE01            CP      1               ; A-1 will set Z if A is also 1
3D              DEC     A               ; A=A-1, Z is set if A is now 0
```

Internally, `CP 1` just does A-1 but discards the result which is why `DEC A` works in this instance.

Compare number

With `CP` it's easy to test for less than (<), equals (=), not equals (!=) or greater-than-equals (>=) because of how the C and Z flags are used:

```
                CP      15                      ; test A against 15
                RET     C                       ; Return if A < 15
                RET     NC                      ; Return if A >= 15
                RET     Z                       ; Return if A = 15
                RET     NZ                      ; Return if A != 15
```

The following shows how to get the other two tests, Less-Than-Equals (<=) and Greater-Than(>):

A <= n

This is a simple one. As `CP` tests against A-n then if A=N then Z is set but if A < n then C is set.

```
                CP      15                      ; test for A<=15
                RET     C                       ; Return if A<15
                RET     Z                       ; Return if A=15
```

To optimise this we should test against n+1 instead. Doing this means we can just use the Carry flag as it would be set when A < n+1:

```
                CP      15+1                    ; test for A<16
                RET     C                       ; Return if A<16
```

A > n

This is the opposite problem. Here Carry is clear when A>=n, so to get A>n we first need to test for equals using the Z flag and if that's not set then check for the Carry flag to be clear:

```
                CP      15                      ; test for A>15
                JR      Z, skip                 ; Skip if A=15
                RET     NC                      ; Return if A>=15
.skip                                           ; Continue as A was <= 15
```

Like the previous example, this can be optimised simply by adding 1 and then testing for A >= (n+1) instead:

```
                CP      15+1                    ; test for A>=16
                RET     NC                      ; Return if A>=16
```

Wasteful use of CP

It's easy to forget that some instructions set the flags based on the result so frequently you do not need to use `CP` to test for a condition when the result is already known:

Here we check for bit 1 of A is set and if it is we exit the subroutine:

```
E601            AND     1                       ; A=A AND 0x01
FE01            CP      1                       ; Is A set to 1
C8              RET     Z                       ; Return is A is now 1
```

Here the `CP` isn't required as `AND` will set Z if A=0, so we can remove the `CP` and use `NZ` instead saving 2 bytes:

```
E601            AND     1                       ; A=A AND 0x01
C8              RET     NZ                      ; Return as A is now 1
```

Testing bits

Testing Bit 0 of A

The standard method of testing if bit 0 of A is set is to use `BIT 0,A`:

```
CB47            BIT     0,A                     ; Test if BIT 0 is set
C8              RET     NZ                      ; Return as bit 0 of A was set
```

If we don't need A afterwards then we can optimise this by using a right shift instead:

```
1F              RRA                             ; Shift A right 1 bit, C=original bit 0
C8              RET     C                       ; Return as bit 0 of A was set
```

This works as we just shifted bit 0 into the Carry Flag and we save an additional byte in the process.

Using `RRA` would be faster & saves 1 byte, but it destroys A. If you need to keep A intact then keep the BIT instruction.

Testing Bit 7 of A

Just like testing bit 0, with bit 7 we can do the same but shifting right instead. So rather than using `BIT 7,A` like:

```
CB7F            BIT     7,A                     ; Test if BIT 7 is set
C8              RET     NZ                      ; Return as bit 7 of A was set
```

We can just use `RLA` and test the Carry flag:

```
17              RLA                             ; Shift A left 1 bit, C=original bit 7
C8              RET     C                       ; Return as bit 7 of A was set
```

The downside of this is it destroys the contents of A.

5.3 - Math

Optimising mathematics

Basic Arithmetic

A=-B

A simple one, we want to set A to be -B.

The logical way is to load A with B then negate it:

```
78              LD      A,B                     ; Set A to B
ED44            NEG                             ; Negate A to get A=-B
```

But a quicker and shorter way is:

```
AF              XOR     A                       ; A=0
90              SUB     B                       ; A=0-B = -B
```

5.4 - Bit Shifting

Optimising bit shifting

Bit shifting, be it rotating left or right is so common it's easy to create slow code if you are not careful.

Shift BC, DE or HL left one bit

This is a 16 bit shift left operation. The first thought would be, especially if you have a 6502 background like myself, is to shift L left 1 bit, clearing bit 0 with carry set to the original bit 7 state, then shift H left 1 bit pulling in carry into bit 0:

```
CB25            SLA     L               ; Shift L left, set bit 0 to 0
CB14            RL      H               ; Shift H left, set bit 0 to original bit 7 from L
```

However any shift left operation is the same as multiplying the value by 2 or just adding to itself, and the Z80 has a single byte operation to do this.

```
29              ADD     HL,HL           ; Shift HL left 1 bit
```

The same applies for BC or DE. If you need to shift a 16-bit register left one bit then always use `ADD`.

Shift 8-bit register left one bit

This might seem odd but the same optimisation can be done for any of the 8-bit registers. You can either use `SLA` or you can just add the register to itself.

```
                                        ; Shift A left one bit, set bit 0 to 0
CB27            SLA     A               ; 2 bytes 8 t-states
87              ADD     A,A             ; 1 byte 4 t-states
```

Here we can halve both the code size and the time taken to perform the shift.

The downside with `ADD` is that the original bit 7 of the register is lost. `SLA` will preserve it in the Carry flag.

Other than that it's identical, with Z set if the register is now 0 and S set if the new bit 7 is set.

6 - reference

6.1 - Instruction List by name

ADC A,(HL)	8E	*AND IXl*	*DDA5*	BIT 2,(IX+d)	DDCBnn56	BIT 4,A	CB67nn
ADC A,(IX+d)	DD8Enn	*AND IYh*	*FDA4*	*BIT 2,(IX+d)*	*DDCBnn57*	BIT 4,B	CB60nn
ADC A,(IY+d)	FD8Enn	*AND IYl*	*FDA5*	*BIT 2,(IY+d)*	*FDCBnn50*	BIT 4,C	CB61nn
ADC A,A	8F	BIT 0,(HL)	CB46nn	*BIT 2,(IY+d)*	*FDCBnn51*	BIT 4,D	CB62nn
ADC A,B	88	*BIT 0,(IX+d)*	*DDCBnn40*	*BIT 2,(IY+d)*	*FDCBnn52*	BIT 4,E	CB63nn
ADC A,C	89	*BIT 0,(IX+d)*	*DDCBnn41*	*BIT 2,(IY+d)*	*FDCBnn53*	BIT 4,H	CB64nn
ADC A,D	8A	*BIT 0,(IX+d)*	*DDCBnn42*	*BIT 2,(IY+d)*	*FDCBnn54*	BIT 4,L	CB65nn
ADC A,E	8B	*BIT 0,(IX+d)*	*DDCBnn43*	*BIT 2,(IY+d)*	*FDCBnn55*	BIT 5,(HL)	CB6Enn
ADC A,H	8C	*BIT 0,(IX+d)*	*DDCBnn44*	BIT 2,(IY+d)	FDCBnn56	*BIT 5,(IX+d)*	*DDCBnn68*
ADC A,IXh	*DD8C*	*BIT 0,(IX+d)*	*DDCBnn45*	*BIT 2,(IY+d)*	*FDCBnn57*	*BIT 5,(IX+d)*	*DDCBnn69*
ADC A,IXl	*DD8D*	BIT 0,(IX+d)	DDCBnn46	BIT 2,A	CB57nn	*BIT 5,(IX+d)*	*DDCBnn6A*
ADC A,IYh	*FD8C*	*BIT 0,(IX+d)*	*DDCBnn47*	BIT 2,B	CB50nn	*BIT 5,(IX+d)*	*DDCBnn6B*
ADC A,IYl	*FD8D*	*BIT 0,(IY+d)*	*FDCBnn40*	BIT 2,C	CB51nn	*BIT 5,(IX+d)*	*DDCBnn6C*
ADC A,L	8D	*BIT 0,(IY+d)*	*FDCBnn41*	BIT 2,D	CB52nn	*BIT 5,(IX+d)*	*DDCBnn6D*
ADC A,n	CEnn	*BIT 0,(IY+d)*	*FDCBnn42*	BIT 2,E	CB53nn	BIT 5,(IX+d)	DDCBnn6E
ADC HL,BC	ED4Ann	*BIT 0,(IY+d)*	*FDCBnn43*	BIT 2,H	CB54nn	*BIT 5,(IX+d)*	*DDCBnn6F*
ADC HL,DE	ED5Ann	*BIT 0,(IY+d)*	*FDCBnn44*	BIT 2,L	CB55nn	*BIT 5,(IY+d)*	*FDCBnn68*
ADC HL,HL	ED6Ann	*BIT 0,(IY+d)*	*FDCBnn45*	BIT 3,(HL)	CB5Enn	*BIT 5,(IY+d)*	*FDCBnn69*
ADC HL,SP	ED7Ann	BIT 0,(IY+d)	FDCBnn46	*BIT 3,(IX+d)*	*DDCBnn58*	*BIT 5,(IY+d)*	*FDCBnn6A*
ADD A,(HL)	86	*BIT 0,(IY+d)*	*FDCBnn47*	*BIT 3,(IX+d)*	*DDCBnn59*	*BIT 5,(IY+d)*	*FDCBnn6B*
ADD A,(IX+d)	DD86nn	BIT 0,A	CB47nn	*BIT 3,(IX+d)*	*DDCBnn5A*	*BIT 5,(IY+d)*	*FDCBnn6C*
ADD A,(IY+d)	FD86nn	BIT 0,B	CB40nn	*BIT 3,(IX+d)*	*DDCBnn5B*	*BIT 5,(IY+d)*	*FDCBnn6D*
ADD A,A	87	BIT 0,C	CB41nn	*BIT 3,(IX+d)*	*DDCBnn5C*	BIT 5,(IY+d)	FDCBnn6E
ADD A,B	80	BIT 0,D	CB42nn	*BIT 3,(IX+d)*	*DDCBnn5D*	*BIT 5,(IY+d)*	*FDCBnn6F*
ADD A,C	81	BIT 0,E	CB43nn	BIT 3,(IX+d)	DDCBnn5E	BIT 5,A	CB6Fnn
ADD A,D	82	BIT 0,H	CB44nn	*BIT 3,(IX+d)*	*DDCBnn5F*	BIT 5,B	CB68nn
ADD A,E	83	BIT 0,L	CB45nn	*BIT 3,(IY+d)*	*FDCBnn58*	BIT 5,C	CB69nn
ADD A,H	84	BIT 1,(HL)	CB4Enn	*BIT 3,(IY+d)*	*FDCBnn59*	BIT 5,D	CB6Ann
ADD A,IXh	*DD84*	*BIT 1,(IX+d)*	*DDCBnn48*	*BIT 3,(IY+d)*	*FDCBnn5A*	BIT 5,E	CB6Bnn
ADD A,IXl	*DD85*	*BIT 1,(IX+d)*	*DDCBnn49*	*BIT 3,(IY+d)*	*FDCBnn5B*	BIT 5,H	CB6Cnn
ADD A,IYh	*FD84*	*BIT 1,(IX+d)*	*DDCBnn4A*	*BIT 3,(IY+d)*	*FDCBnn5C*	BIT 5,L	CB6Dnn
ADD A,IYl	*FD85*	*BIT 1,(IX+d)*	*DDCBnn4B*	*BIT 3,(IY+d)*	*FDCBnn5D*	BIT 6,(HL)	CB76nn
ADD A,L	85	*BIT 1,(IX+d)*	*DDCBnn4C*	BIT 3,(IY+d)	FDCBnn5E	*BIT 6,(IX+d)*	*DDCBnn70*
ADD A,n	C6nn	*BIT 1,(IX+d)*	*DDCBnn4D*	*BIT 3,(IY+d)*	*FDCBnn5F*	*BIT 6,(IX+d)*	*DDCBnn71*
ADD HL,BC	09	BIT 1,(IX+d)	DDCBnn4E	BIT 3,A	CB5Fnn	*BIT 6,(IX+d)*	*DDCBnn72*
ADD HL,DE	19	*BIT 1,(IX+d)*	*DDCBnn4F*	BIT 3,B	CB58nn	*BIT 6,(IX+d)*	*DDCBnn73*
ADD HL,HL	29	*BIT 1,(IY+d)*	*FDCBnn48*	BIT 3,C	CB59nn	*BIT 6,(IX+d)*	*DDCBnn74*
ADD HL,SP	39	*BIT 1,(IY+d)*	*FDCBnn49*	BIT 3,D	CB5Ann	*BIT 6,(IX+d)*	*DDCBnn75*
ADD IX,BC	DD09nn	*BIT 1,(IY+d)*	*FDCBnn4A*	BIT 3,E	CB5Bnn	BIT 6,(IX+d)	DDCBnn76
ADD IX,DE	DD19nn	*BIT 1,(IY+d)*	*FDCBnn4B*	BIT 3,H	CB5Cnn	*BIT 6,(IX+d)*	*DDCBnn77*
ADD IX,IX	DD29nn	*BIT 1,(IY+d)*	*FDCBnn4C*	BIT 3,L	CB5Dnn	*BIT 6,(IY+d)*	*FDCBnn70*
ADD IX,SP	DD39nn	*BIT 1,(IY+d)*	*FDCBnn4D*	BIT 4,(HL)	CB66nn	*BIT 6,(IY+d)*	*FDCBnn71*
ADD IY,BC	FD09nn	BIT 1,(IY+d)	FDCBnn4E	*BIT 4,(IX+d)*	*DDCBnn60*	*BIT 6,(IY+d)*	*FDCBnn72*
ADD IY,DE	FD19nn	*BIT 1,(IY+d)*	*FDCBnn4F*	*BIT 4,(IX+d)*	*DDCBnn61*	*BIT 6,(IY+d)*	*FDCBnn73*
ADD IY,IY	FD29nn	BIT 1,A	CB4Fnn	*BIT 4,(IX+d)*	*DDCBnn62*	*BIT 6,(IY+d)*	*FDCBnn74*
ADD IY,SP	FD39nn	BIT 1,B	CB48nn	*BIT 4,(IX+d)*	*DDCBnn63*	*BIT 6,(IY+d)*	*FDCBnn75*
AND A,(HL)	A6	BIT 1,C	CB49nn	*BIT 4,(IX+d)*	*DDCBnn64*	BIT 6,(IY+d)	FDCBnn76
AND A,(IX+d)	DDA6nn	BIT 1,D	CB4Ann	*BIT 4,(IX+d)*	*DDCBnn65*	*BIT 6,(IY+d)*	*FDCBnn77*
AND A,(IY+d)	FDA6nn	BIT 1,E	CB4Bnn	BIT 4,(IX+d)	DDCBnn66	BIT 6,A	CB77nn
AND A,A	A7	BIT 1,H	CB4Cnn	*BIT 4,(IX+d)*	*DDCBnn67*	BIT 6,B	CB70nn
AND A,B	A0	BIT 1,L	CB4Dnn	*BIT 4,(IY+d)*	*FDCBnn60*	BIT 6,C	CB71nn
AND A,C	A1	BIT 2,(HL)	CB56nn	*BIT 4,(IY+d)*	*FDCBnn61*	BIT 6,D	CB72nn
AND A,D	A2	*BIT 2,(IX+d)*	*DDCBnn50*	*BIT 4,(IY+d)*	*FDCBnn62*	BIT 6,E	CB73nn
AND A,E	A3	*BIT 2,(IX+d)*	*DDCBnn51*	*BIT 4,(IY+d)*	*FDCBnn63*	BIT 6,H	CB74nn
AND A,H	A4	*BIT 2,(IX+d)*	*DDCBnn52*	*BIT 4,(IY+d)*	*FDCBnn64*	BIT 6,L	CB75nn
AND A,L	A5	*BIT 2,(IX+d)*	*DDCBnn53*	*BIT 4,(IY+d)*	*FDCBnn65*	BIT 7,(HL)	CB7Enn
AND A,n	E6nn	*BIT 2,(IX+d)*	*DDCBnn54*	BIT 4,(IY+d)	FDCBnn66	*BIT 7,(IX+d)*	*DDCBnn78*
AND IXh	*DDA4*	*BIT 2,(IX+d)*	*DDCBnn55*	*BIT 4,(IY+d)*	*FDCBnn67*	*BIT 7,(IX+d)*	*DDCBnn79*

BIT 7,(IX+d)	*DDCBnn7A*	DEC DE	1B	JP N,nn	FAnnnn	LD A, n	3Enn
BIT 7,(IX+d)	*DDCBnn7B*	DEC E	1D	JP NC,nn	D2nnnn	*LD A,IXh*	*DD7C*
BIT 7,(IX+d)	*DDCBnn7C*	DEC H	25	JP NZ,nn	C2nnnn	*LD A,IXl*	*DD7D*
BIT 7,(IX+d)	*DDCBnn7D*	DEC HL	2B	JP P,nn	F2nnnn	*LD A,IYh*	*FD7C*
BIT 7,(IX+d)	DDCBnn7E	DEC IX	DD2Bnn	JP PE,nn	EAnnnn	*LD A,IYl*	*FD7D*
BIT 7,(IX+d)	*DDCBnn7F*	*DEC IXh*	*DD25*	JP PO,nn	E2nnnn	LD B, (HL)	46
BIT 7,(IY+d)	*FDCBnn78*	*DEC IXl*	*DD2D*	JP Z,nn	CAnnnn	LD B, (IX+d)	DD46nn
BIT 7,(IY+d)	*FDCBnn79*	DEC IY	FD2Bnn	JP nn	C3nnnn	LD B, (IY+d)	FD46nn
BIT 7,(IY+d)	*FDCBnn7A*	*DEC IYh*	*FD25*	JR C,e	38nn	LD B, A	47
BIT 7,(IY+d)	*FDCBnn7B*	*DEC IYl*	*FD2D*	JR NC,e	30nn	LD B, B	40
BIT 7,(IY+d)	*FDCBnn7C*	DEC L	2D	JR NZ,e	20nn	LD B, C	41
BIT 7,(IY+d)	*FDCBnn7D*	DEC SP	3B	JR Z,e	28nn	LD B, D	42
BIT 7,(IY+d)	FDCBnn7E	DI	F3	JR e	18nn	LD B, E	43
BIT 7,(IY+d)	*FDCBnn7F*	DJNZ e	10nn	LD (BC), A	02	LD B, H	44
BIT 7,A	CB7Fnn	EI	FB	LD (DE), A	12	LD B, L	45
BIT 7,B	CB78nn	EX (SP), HL	E3	LD (HL), A	77	LD B, n	06nn
BIT 7,C	CB79nn	EX (SP), IX	DDE3nn	LD (HL), B	70	*LD B,IXh*	*DD44*
BIT 7,D	CB7Ann	EX (SP), IY	FDE3nn	LD (HL), C	71	*LD B,IXl*	*DD45*
BIT 7,E	CB7Bnn	EX AF, AF'	08	LD (HL), D	72	*LD B,IYh*	*FD44*
BIT 7,H	CB7Cnn	EX DE, HL	EB	LD (HL), E	73	*LD B,IYl*	*FD45*
BIT 7,L	CB7Dnn	EXX	D9	LD (HL), H	74	LD BC, (nn)	ED4Bnnnn
CALL C,nn	DCnnnn	HALT	76	LD (HL), L	75	LD BC, nn	01nnnn
CALL N,nn	FCnnnn	IM0	ED46nn	LD (HL), n	36nn	LD C, (HL)	4E
CALL NC,nn	D4nnnn	IM1	ED56nn	LD (IX+d), A	DD77nn	LD C, (IX+d)	DD4Enn
CALL NZ,nn	C4nnnn	IM2	ED5Enn	LD (IX+d), B	DD70nn	LD C, (IY+d)	FD4Enn
CALL P,nn	F4nnnn	IN A,(C)	ED7Bnn	LD (IX+d), C	DD71nn	LD C, A	4F
CALL PE,nn	ECnnnn	IN A,(n)	DBnn	LD (IX+d), D	DD72nn	LD C, B	48
CALL PO,nn	E4nnnn	IN B,(C)	ED40nn	LD (IX+d), E	DD73nn	LD C, C	49
CALL Z,nn	CCnnnn	IN C,(C)	ED48nn	LD (IX+d), H	DD74nn	LD C, D	4A
CALL nn	CDnnnn	IN D,(C)	ED50nn	LD (IX+d), L	DD75nn	LD C, E	4B
CCF	3F	IN E,(C)	ED58nn	LD (IX+d), n	DD36nnnn	LD C, H	4C
CCF	3F	*IN F,(C)*	*ED70nn*	LD (IY+d), A	FD77nn	LD C, L	4D
CP (HL)	BE	IN H,(C)	ED60nn	LD (IY+d), B	FD70nn	LD C, n	0Enn
CP (IX+d)	DDBEnn	IN L,(C)	ED68nn	LD (IY+d), C	FD71nn	*LD C,IXh*	*DD4C*
CP (IY+d)	FDBEnn	INC (HL)	34	LD (IY+d), D	FD72nn	*LD C,IXl*	*DD4D*
CP A	BF	INC (IX+d)	DD34nn	LD (IY+d), E	FD73nn	*LD C,IYh*	*FD4C*
CP B	B8	INC (IY+d)	FD34nn	LD (IY+d), H	FD74nn	*LD C,IYl*	*FD4D*
CP C	B9	INC A	3C	LD (IY+d), L	FD75nn	LD D, (HL)	56
CP D	BA	INC B	04	LD (IY+d), n	FD36nnnn	LD D, (IX+d)	DD56nn
CP E	BB	INC BC	03	LD (nn), A	32nnnn	LD D, (IY+d)	FD56nn
CP H	BC	INC C	0C	LD (nn), BC	ED43nnnn	LD D, A	57
CP IXh	*DDBC*	INC D	14	LD (nn), DE	ED53nnnn	LD D, B	50
CP IXl	*DDBD*	INC DE	13	LD (nn), HL	22nnnn	LD D, C	51
CP IYh	*FDBC*	INC E	1C	LD (nn), HL	ED63nnnn	LD D, D	52
CP IYl	*FDBD*	INC H	24	LD (nn), IX	DD22nnnn	LD D, E	53
CP L	BD	INC HL	23	LD (nn), IY	FD22nnnn	LD D, H	54
CP n	FEnn	INC IX	DD23nn	LD (nn), SP	ED73nnnn	LD D, L	55
CPD	EDA9nn	*INC IXh*	*DD24*	LD A, (BC)	0A	LD D, n	16nn
CPDR	EDB9nn	*INC IXl*	*DD2C*	LD A, (DE)	1A	*LD D,IXh*	*DD54*
CPI	EDA1nn	INC IY	FD23nn	LD A, (HL)	7E	*LD D,IXl*	*DD55*
CPIR	EDB1nn	*INC IYh*	*FD24*	LD A, (IX+d)	DD7Enn	*LD D,IYh*	*FD54*
CPL	2F	*INC IYl*	*FD2C*	LD A, (IY+d)	FD7Enn	*LD D,IYl*	*FD55*
CPL	2F	INC L	2C	LD A, (nn)	3Annnn	LD DE, (nn)	ED5Bnnnn
DAA	27	INC SP	33	LD A, A	7F	LD DE, nn	11nnnn
DEC (HL)	35	IND	EDAAnn	LD A, B	78	LD E, (HL)	5E
DEC (IX+d)	DD35nn	INDR	EDBAnn	LD A, C	79	LD E, (IX+d)	DD5Enn
DEC (IY+d)	FD35nn	INI	EDA2nn	LD A, D	7A	LD E, (IY+d)	FD5Enn
DEC A	3D	INIR	EDB2nn	LD A, E	7B	LD E, A	5F
DEC B	05	JP (HL)	E9	LD A, H	7C	LD E, B	58
DEC BC	0B	JP (IX)	DDE9nn	LD A, I	ED57nn	LD E, C	59
DEC C	0D	JP (IY)	FDE9nn	LD A, L	7D	LD E, D	5A
DEC D	15	JP C,nn	DAnnnn	LD A, R	ED5Fnn	LD E, E	5B

LD E, H	5C	LD L, B	68	RES 0,(IY+d)	FDCBnn86	RES 6,B	CBB0nn
LD E, L	5D	LD L, C	69	RES 0,A	CB87nn	RES 6,C	CBB1nn
LD E, n	1Enn	LD L, D	6A	RES 0,B	CB80nn	RES 6,D	CBB2nn
LD E,IXh	*DD5C*	LD L, E	6B	RES 0,C	CB81nn	RES 6,E	CBB3nn
LD E,IXl	*DD5D*	LD L, H	6C	RES 0,D	CB82nn	RES 6,H	CBB4nn
LD E,IYh	*FD5C*	LD L, L	6D	RES 0,E	CB83nn	RES 6,L	CBB5nn
LD E,IYl	*FD5D*	LD L, n	2Enn	RES 0,H	CB84nn	RES 7,(HL)	CBBEnn
LD H, (HL)	66	LD R, A	ED4Fnn	RES 0,L	CB85nn	RES 7,(IX+d)	DDCBnnBE
LD H, (IX+d)	DD66nn	LD SP, (nn)	ED7Bnnnn	RES 1,(HL)	CB8Enn	RES 7,(IY+d)	FDCBnnBE
LD H, (IY+d)	FD66nn	LD SP, HL	F9	RES 1,(IX+d)	DDCBnn8E	RES 7,A	CBBFnn
LD H, A	67	LD SP, IX	DDF9nn	RES 1,(IY+d)	FDCBnn8E	RES 7,B	CBB8nn
LD H, B	60	LD SP, IY	FDF9nn	RES 1,A	CB8Fnn	RES 7,C	CBB9nn
LD H, C	61	LD SP, nn	31nnnn	RES 1,B	CB88nn	RES 7,D	CBBAnn
LD H, D	62	LDD	EDA8nn	RES 1,C	CB89nn	RES 7,E	CBBBnn
LD H, E	63	LDDR	EDB8nn	RES 1,D	CB8Ann	RES 7,H	CBBCnn
LD H, H	64	LDI	EDA0nn	RES 1,E	CB8Bnn	RES 7,L	CBBDnn
LD H, L	65	LDIR	EDB0nn	RES 1,H	CB8Cnn	*RES A,0,(IX+nn)*	*DDCBnn87*
LD H, n	26nn	NEG	ED44nn	RES 1,L	CB8Dnn	*RES A,0,(IY+nn)*	*FDCBnn87*
LD HL, (nn)	2Annnn	NEG	ED44nn	RES 2,(HL)	CB96nn	*RES A,1,(IX+nn)*	*DDCBnn8F*
LD HL, (nn)	ED6Bnnnn	NOP	00	RES 2,(IX+d)	DDCBnn96	*RES A,1,(IY+nn)*	*FDCBnn8F*
LD HL, nn	21nnnn	OR A,(HL)	B6	RES 2,(IY+d)	FDCBnn96	*RES A,2,(IX+nn)*	*DDCBnn97*
LD I, A	ED47nn	OR A,(IX+d)	DDB6nn	RES 2,A	CB97nn	*RES A,2,(IY+nn)*	*FDCBnn97*
LD IX, (nn)	DD2Annnn	OR A,(IY+d)	FDB6nn	RES 2,B	CB90nn	*RES A,3,(IX+nn)*	*DDCBnn9F*
LD IX, nn	DD21nnnn	OR A,A	B7	RES 2,C	CB91nn	*RES A,3,(IY+nn)*	*FDCBnn9F*
LD IXh,A	*DD67*	OR A,B	B0	RES 2,D	CB92nn	*RES A,4,(IX+nn)*	*DDCBnnA7*
LD IXh,B	*DD60*	OR A,C	B1	RES 2,E	CB93nn	*RES A,4,(IY+nn)*	*FDCBnnA7*
LD IXh,C	*DD61*	OR A,D	B2	RES 2,H	CB94nn	*RES A,5,(IX+nn)*	*DDCBnnAF*
LD IXh,D	*DD62*	OR A,E	B3	RES 2,L	CB95nn	*RES A,5,(IY+nn)*	*FDCBnnAF*
LD IXh,E	*DD63*	OR A,H	B4	RES 3,(HL)	CB9Enn	*RES A,6,(IX+nn)*	*DDCBnnB7*
LD IXh,IHh	*DD64*	OR A,L	B5	RES 3,(IX+d)	DDCBnn9E	*RES A,6,(IY+nn)*	*FDCBnnB7*
LD IXh,IHl	*DD65*	OR A,n	F6nn	RES 3,(IY+d)	FDCBnn9E	*RES A,7,(IX+nn)*	*DDCBnnBF*
LD IXh,n	*DD26nn*	*OR IXh*	*DDB4*	RES 3,A	CB9Fnn	*RES A,7,(IY+nn)*	*FDCBnnBF*
LD IXl,A	*DD6F*	*OR IXl*	*DDB5*	RES 3,B	CB98nn	*RES B,0,(IX+nn)*	*DDCBnn80*
LD IXl,B	*DD68*	*OR IYh*	*FDB4*	RES 3,C	CB99nn	*RES B,0,(IY+nn)*	*FDCBnn80*
LD IXl,C	*DD69*	*OR IYl*	*FDB5*	RES 3,D	CB9Ann	*RES B,1,(IX+nn)*	*DDCBnn88*
LD IXl,D	*DD6A*	OUT (C),A	ED79nn	RES 3,E	CB9Bnn	*RES B,1,(IY+nn)*	*FDCBnn88*
LD IXl,E	*DD6B*	OUT (C),B	ED41nn	RES 3,H	CB9Cnn	*RES B,2,(IX+nn)*	*DDCBnn90*
LD IXl,IHh	*DD6C*	OUT (C),C	ED49nn	RES 3,L	CB9Dnn	*RES B,2,(IY+nn)*	*FDCBnn90*
LD IXl,IHl	*DD6D*	OUT (C),D	ED51nn	RES 4,(HL)	CBA6nn	*RES B,3,(IX+nn)*	*DDCBnn98*
LD IXl,n	*DD2Enn*	OUT (C),E	ED59nn	RES 4,(IX+d)	DDCBnnA6	*RES B,3,(IY+nn)*	*FDCBnn98*
LD IY, (nn)	FD2Annnn	*OUT (C),F*	*ED71nn*	RES 4,(IY+d)	FDCBnnA6	*RES B,4,(IX+nn)*	*DDCBnnA0*
LD IY, nn	FD21nnnn	OUT (C),H	ED61nn	RES 4,A	CBA7nn	*RES B,4,(IY+nn)*	*FDCBnnA0*
LD IYh,A	*FD67*	OUT (C),L	ED69nn	RES 4,B	CBA0nn	*RES B,5,(IX+nn)*	*DDCBnnA8*
LD IYh,B	*FD60*	OUT (n),A	D3nn	RES 4,C	CBA1nn	*RES B,5,(IY+nn)*	*FDCBnnA8*
LD IYh,C	*FD61*	OUTD	EDABnn	RES 4,D	CBA2nn	*RES B,6,(IX+nn)*	*DDCBnnB0*
LD IYh,D	*FD62*	OUTDR	EDBBnn	RES 4,E	CBA3nn	*RES B,6,(IY+nn)*	*FDCBnnB0*
LD IYh,E	*FD63*	OUTI	EDA3nn	RES 4,H	CBA4nn	*RES B,7,(IX+nn)*	*DDCBnnB8*
LD IYh,IHh	*FD64*	OUTIR	EDB3nn	RES 4,L	CBA5nn	*RES B,7,(IY+nn)*	*FDCBnnB8*
LD IYh,IHl	*FD65*	POP AF	F1	RES 5,(HL)	CBAEnn	*RES C,0,(IX+nn)*	*DDCBnn81*
LD IYh,n	*FD26nn*	POP BC	C1	RES 5,(IX+d)	DDCBnnAE	*RES C,0,(IY+nn)*	*FDCBnn81*
LD IYl,A	*FD6F*	POP DE	D1	RES 5,(IY+d)	FDCBnnAE	*RES C,1,(IX+nn)*	*DDCBnn89*
LD IYl,B	*FD68*	POP HL	E1	RES 5,A	CBAFnn	*RES C,1,(IY+nn)*	*FDCBnn89*
LD IYl,C	*FD69*	POP IX	DDE1nn	RES 5,B	CBA8nn	*RES C,2,(IX+nn)*	*DDCBnn91*
LD IYl,D	*FD6A*	POP IY	FDE1nn	RES 5,C	CBA9nn	*RES C,2,(IY+nn)*	*FDCBnn91*
LD IYl,E	*FD6B*	PUSH AF	F5	RES 5,D	CBAAnn	*RES C,3,(IX+nn)*	*DDCBnn99*
LD IYl,IHh	*FD6C*	PUSH BC	C5	RES 5,E	CBABnn	*RES C,3,(IY+nn)*	*FDCBnn99*
LD IYl,IHl	*FD6D*	PUSH DE	D5	RES 5,H	CBACnn	*RES C,4,(IX+nn)*	*DDCBnnA1*
LD IYl,n	*FD2Enn*	PUSH HL	E5	RES 5,L	CBADnn	*RES C,4,(IY+nn)*	*FDCBnnA1*
LD L, (HL)	6E	PUSH IX	DDE5nn	RES 6,(HL)	CBB6nn	*RES C,5,(IX+nn)*	*DDCBnnA9*
LD L, (IX+d)	DD6Enn	PUSH IY	FDE5nn	RES 6,(IX+d)	DDCBnnB6	*RES C,5,(IY+nn)*	*FDCBnnA9*
LD L, (IY+d)	FD6Enn	RES 0,(HL)	CB86nn	RES 6,(IY+d)	FDCBnnB6	*RES C,6,(IX+nn)*	*DDCBnnB1*
LD L, A	6F	RES 0,(IX+d)	DDCBnn86	RES 6,A	CBB7nn	*RES C,6,(IY+nn)*	*FDCBnnB1*

RES C,7,(IX+nn)	*DDCBnnB9*	*RES L,6,(IX+nn)*	*DDCBnnB5*	*RLC L,(IX+d)*	*DDCBnn05*
RES C,7,(IY+nn)	*FDCBnnB9*	*RES L,6,(IY+nn)*	*FDCBnnB5*	*RLC L,(IY+d)*	*FDCBnn05*
RES D,0,(IX+nn)	*DDCBnn82*	*RES L,7,(IX+nn)*	*DDCBnnBD*	RLCA	07
RES D,0,(IY+nn)	*FDCBnn82*	*RES L,7,(IY+nn)*	*FDCBnnBD*	RLD (HL)	ED6Fnn
RES D,1,(IX+nn)	*DDCBnn8A*	RET	C9	RR (HL)	CB1Enn
RES D,1,(IY+nn)	*FDCBnn8A*	RET C	D8	RR (IX+d)	DDCBnn1E
RES D,2,(IX+nn)	*DDCBnn92*	RET N	F8	RR (IY+d)	FDCBnn1E
RES D,2,(IY+nn)	*FDCBnn92*	RET NC	D0	RR A	CB1Fnn
RES D,3,(IX+nn)	*DDCBnn9A*	RET NZ	C0	*RR A,(IX+d)*	*DDCBnn1F*
RES D,3,(IY+nn)	*FDCBnn9A*	RET P	F0	*RR A,(IY+d)*	*FDCBnn1F*
RES D,4,(IX+nn)	*DDCBnnA2*	RET PE	E8	RR B	CB18nn
RES D,4,(IY+nn)	*FDCBnnA2*	RET PO	E0	*RR B,(IX+d)*	*DDCBnn18*
RES D,5,(IX+nn)	*DDCBnnAA*	RET Z	C8	*RR B,(IY+d)*	*FDCBnn18*
RES D,5,(IY+nn)	*FDCBnnAA*	RETI	ED4Dnn	RR C	CB19nn
RES D,6,(IX+nn)	*DDCBnnB2*	RETN	ED45nn	*RR C,(IX+d)*	*DDCBnn19*
RES D,6,(IY+nn)	*FDCBnnB2*	RL (HL)	CB16nn	*RR C,(IY+d)*	*FDCBnn19*
RES D,7,(IX+nn)	*DDCBnnBA*	RL (IX+d)	DDCBnn16	RR D	CB1Ann
RES D,7,(IY+nn)	*FDCBnnBA*	RL (IY+d)	FDCBnn16	*RR D,(IX+d)*	*DDCBnn1A*
RES E,0,(IX+nn)	*DDCBnn83*	RL A	CB17nn	*RR D,(IY+d)*	*FDCBnn1A*
RES E,0,(IY+nn)	*FDCBnn83*	*RL A,(IX+d)*	*DDCBnn17*	RR E	CB1Bnn
RES E,1,(IX+nn)	*DDCBnn8B*	*RL A,(IY+d)*	*FDCBnn17*	*RR E,(IX+d)*	*DDCBnn1B*
RES E,1,(IY+nn)	*FDCBnn8B*	RL B	CB10nn	*RR E,(IY+d)*	*FDCBnn1B*
RES E,2,(IX+nn)	*DDCBnn93*	*RL B,(IX+d)*	*DDCBnn10*	RR H	CB1Cnn
RES E,2,(IY+nn)	*FDCBnn93*	*RL B,(IY+d)*	*FDCBnn10*	*RR H,(IX+d)*	*DDCBnn1C*
RES E,3,(IX+nn)	*DDCBnn9B*	RL C	CB11nn	*RR H,(IY+d)*	*FDCBnn1C*
RES E,3,(IY+nn)	*FDCBnn9B*	*RL C,(IX+d)*	*DDCBnn11*	RR L	CB1Dnn
RES E,4,(IX+nn)	*DDCBnnA3*	*RL C,(IY+d)*	*FDCBnn11*	*RR L,(IX+d)*	*DDCBnn1D*
RES E,4,(IY+nn)	*FDCBnnA3*	RL D	CB12nn	*RR L,(IY+d)*	*FDCBnn1D*
RES E,5,(IX+nn)	*DDCBnnAB*	*RL D,(IX+d)*	*DDCBnn12*	RRA	1F
RES E,5,(IY+nn)	*FDCBnnAB*	*RL D,(IY+d)*	*FDCBnn12*	RRC (HL)	CB0Enn
RES E,6,(IX+nn)	*DDCBnnB3*	RL E	CB13nn	RRC (IX+d)	DDCBnn0E
RES E,6,(IY+nn)	*FDCBnnB3*	*RL E,(IX+d)*	*DDCBnn13*	RRC (IY+d)	FDCBnn0E
RES E,7,(IX+nn)	*DDCBnnBB*	*RL E,(IY+d)*	*FDCBnn13*	RRC A	CB0Fnn
RES E,7,(IY+nn)	*FDCBnnBB*	RL H	CB14nn	*RRC A,(IX+d)*	*DDCBnn0F*
RES H,0,(IX+nn)	*DDCBnn84*	*RL H,(IX+d)*	*DDCBnn14*	*RRC A,(IY+d)*	*FDCBnn0F*
RES H,0,(IY+nn)	*FDCBnn84*	*RL H,(IY+d)*	*FDCBnn14*	RRC B	CB08nn
RES H,1,(IX+nn)	*DDCBnn8C*	RL L	CB15nn	*RRC B,(IX+d)*	*DDCBnn08*
RES H,1,(IY+nn)	*FDCBnn8C*	*RL L,(IX+d)*	*DDCBnn15*	*RRC B,(IY+d)*	*FDCBnn08*
RES H,2,(IX+nn)	*DDCBnn94*	*RL L,(IY+d)*	*FDCBnn15*	RRC C	CB09nn
RES H,2,(IY+nn)	*FDCBnn94*	RLA	17	*RRC C,(IX+d)*	*DDCBnn09*
RES H,3,(IX+nn)	*DDCBnn9C*	RLC (HL)	CB06nn	*RRC C,(IY+d)*	*FDCBnn09*
RES H,3,(IY+nn)	*FDCBnn9C*	RLC (IX+d)	DDCBnn06	RRC D	CB0Ann
RES H,4,(IX+nn)	*DDCBnnA4*	RLC (IY+d)	FDCBnn06	*RRC D,(IX+d)*	*DDCBnn0A*
RES H,4,(IY+nn)	*FDCBnnA4*	RLC A	CB07nn	*RRC D,(IY+d)*	*FDCBnn0A*
RES H,5,(IX+nn)	*DDCBnnAC*	*RLC A,(IX+d)*	*DDCBnn07*	RRC E	CB0Bnn
RES H,5,(IY+nn)	*FDCBnnAC*	*RLC A,(IY+d)*	*FDCBnn07*	*RRC E,(IX+d)*	*DDCBnn0B*
RES H,6,(IX+nn)	*DDCBnnB4*	RLC B	CB00nn	*RRC E,(IY+d)*	*FDCBnn0B*
RES H,6,(IY+nn)	*FDCBnnB4*	*RLC B,(IX+d)*	*DDCBnn00*	RRC H	CB0Cnn
RES H,7,(IX+nn)	*DDCBnnBC*	*RLC B,(IY+d)*	*FDCBnn00*	*RRC H,(IX+d)*	*DDCBnn0C*
RES H,7,(IY+nn)	*FDCBnnBC*	RLC C	CB01nn	*RRC H,(IY+d)*	*FDCBnn0C*
RES L,0,(IX+nn)	*DDCBnn85*	*RLC C,(IX+d)*	*DDCBnn01*	RRC L	CB0Dnn
RES L,0,(IY+nn)	*FDCBnn85*	*RLC C,(IY+d)*	*FDCBnn01*	*RRC L,(IX+d)*	*DDCBnn0D*
RES L,1,(IX+nn)	*DDCBnn8D*	RLC D	CB02nn	*RRC L,(IY+d)*	*FDCBnn0D*
RES L,1,(IY+nn)	*FDCBnn8D*	*RLC D,(IX+d)*	*DDCBnn02*	RRCA	0F
RES L,2,(IX+nn)	*DDCBnn95*	*RLC D,(IY+d)*	*FDCBnn02*	RRD (HL)	ED67nn
RES L,2,(IY+nn)	*FDCBnn95*	RLC E	CB03nn	RST 0	C7
RES L,3,(IX+nn)	*DDCBnn9D*	*RLC E,(IX+d)*	*DDCBnn03*	RST 1	CF
RES L,3,(IY+nn)	*FDCBnn9D*	*RLC E,(IY+d)*	*FDCBnn03*	RST 2	D7
RES L,4,(IX+nn)	*DDCBnnA5*	RLC H	CB04nn	RST 3	DF
RES L,4,(IY+nn)	*FDCBnnA5*	*RLC H,(IX+d)*	*DDCBnn04*	RST 4	E7
RES L,5,(IX+nn)	*DDCBnnAD*	*RLC H,(IY+d)*	*FDCBnn04*	RST 5	EF
RES L,5,(IY+nn)	*FDCBnnAD*	RLC L	CB05nn	RST 6	F7
RST 7	FF				
SBC A,(HL)	9E				
SBC A,(IX+d)	DD9Enn				
SBC A,(IY+d)	FD9Enn				
SBC A,A	9F				
SBC A,B	98				
SBC A,C	99				
SBC A,D	9A				
SBC A,E	9B				
SBC A,H	9C				
SBC A,IXh	*DD9C*				
SBC A,IXl	*DD9D*				
SBC A,IYh	*FD9C*				
SBC A,IYl	*FD9D*				
SBC A,L	9D				
SBC A,n	DEnn				
SBC HL,BC	ED42nn				
SBC HL,DE	ED52nn				
SBC HL,HL	ED62nn				
SBC HL,SP	ED72nn				
SCF	37				
SET 0,(HL)	CBC6nn				
SET 0,(IX+d)	DDCBnnC6				
SET 0,(IY+d)	FDCBnnC6				
SET 0,A	CBC7nn				
SET 0,B	CBC0nn				
SET 0,C	CBC1nn				
SET 0,D	CBC2nn				
SET 0,E	CBC3nn				
SET 0,H	CBC4nn				
SET 0,L	CBC5nn				
SET 1,(HL)	CBCEnn				
SET 1,(IX+d)	DDCBnnCE				
SET 1,(IY+d)	FDCBnnCE				
SET 1,A	CBCFnn				
SET 1,B	CBC8nn				
SET 1,C	CBC9nn				
SET 1,D	CBCAnn				
SET 1,E	CBCBnn				
SET 1,H	CBCCnn				
SET 1,L	CBCDnn				
SET 2,(HL)	CBD6nn				
SET 2,(IX+d)	DDCBnnD6				
SET 2,(IY+d)	FDCBnnD6				
SET 2,A	CBD7nn				
SET 2,B	CBD0nn				
SET 2,C	CBD1nn				
SET 2,D	CBD2nn				
SET 2,E	CBD3nn				
SET 2,H	CBD4nn				
SET 2,L	CBD5nn				
SET 3,(HL)	CBDEnn				
SET 3,(IX+d)	DDCBnnDE				
SET 3,(IY+d)	FDCBnnDE				
SET 3,A	CBDFnn				
SET 3,B	CBD8nn				
SET 3,C	CBD9nn				
SET 3,D	CBDAnn				
SET 3,E	CBDBnn				
SET 3,H	CBDCnn				
SET 3,L	CBDDnn				
SET 4,(HL)	CBE6nn				

SET 4,(IX+d) DDCBnnE6
SET 4,(IY+d) FDCBnnE6
SET 4,A CBE7nn
SET 4,B CBE0nn
SET 4,C CBE1nn
SET 4,D CBE2nn
SET 4,E CBE3nn
SET 4,H CBE4nn
SET 4,L CBE5nn
SET 5,(HL) CBEEnn
SET 5,(IX+d) DDCBnnEE
SET 5,(IY+d) FDCBnnEE
SET 5,A CBEFnn
SET 5,B CBE8nn
SET 5,C CBE9nn
SET 5,D CBEAnn
SET 5,E CBEBnn
SET 5,H CBECnn
SET 5,L CBEDnn
SET 6,(HL) CBF6nn
SET 6,(IX+d) DDCBnnF6
SET 6,(IY+d) FDCBnnF6
SET 6,A CBF7nn
SET 6,B CBF0nn
SET 6,C CBF1nn
SET 6,D CBF2nn
SET 6,E CBF3nn
SET 6,H CBF4nn
SET 6,L CBF5nn
SET 7,(HL) CBFEnn
SET 7,(IX+d) DDCBnnFE
SET 7,(IY+d) FDCBnnFE
SET 7,A CBFFnn
SET 7,B CBF8nn
SET 7,C CBF9nn
SET 7,D CBFAnn
SET 7,E CBFBnn
SET 7,H CBFCnn
SET 7,L CBFDnn
SET A,0,(IX+nn) DDCBnnC7
SET A,0,(IY+nn) FDCBnnC7
SET A,1,(IX+nn) DDCBnnCF
SET A,1,(IY+nn) FDCBnnCF
SET A,2,(IX+nn) DDCBnnD7
SET A,2,(IY+nn) FDCBnnD7
SET A,3,(IX+nn) DDCBnnDF
SET A,3,(IY+nn) FDCBnnDF
SET A,4,(IX+nn) DDCBnnE7
SET A,4,(IY+nn) FDCBnnE7
SET A,5,(IX+nn) DDCBnnEF
SET A,5,(IY+nn) FDCBnnEF
SET A,6,(IX+nn) DDCBnnF7
SET A,6,(IY+nn) FDCBnnF7
SET A,7,(IX+nn) DDCBnnFF
SET A,7,(IY+nn) FDCBnnFF
SET B,0,(IX+nn) DDCBnnC0
SET B,0,(IY+nn) FDCBnnC0
SET B,1,(IX+nn) DDCBnnC8
SET B,1,(IY+nn) FDCBnnC8
SET B,2,(IX+nn) DDCBnnD0
SET B,2,(IY+nn) FDCBnnD0
SET B,3,(IX+nn) DDCBnnD8
SET B,3,(IY+nn) FDCBnnD8
SET B,4,(IX+nn) DDCBnnE0
SET B,4,(IY+nn) FDCBnnE0
SET B,5,(IX+nn) DDCBnnE8
SET B,5,(IY+nn) FDCBnnE8
SET B,6,(IX+nn) DDCBnnF0
SET B,6,(IY+nn) FDCBnnF0
SET B,7,(IX+nn) DDCBnnF8
SET B,7,(IY+nn) FDCBnnF8
SET C,0,(IX+nn) DDCBnnC1
SET C,0,(IY+nn) FDCBnnC1
SET C,1,(IX+nn) DDCBnnC9
SET C,1,(IY+nn) FDCBnnC9
SET C,2,(IX+nn) DDCBnnD1
SET C,2,(IY+nn) FDCBnnD1
SET C,3,(IX+nn) DDCBnnD9
SET C,3,(IY+nn) FDCBnnD9
SET C,4,(IX+nn) DDCBnnE1
SET C,4,(IY+nn) FDCBnnE1
SET C,5,(IX+nn) DDCBnnE9
SET C,5,(IY+nn) FDCBnnE9
SET C,6,(IX+nn) DDCBnnF1
SET C,6,(IY+nn) FDCBnnF1
SET C,7,(IX+nn) DDCBnnF9
SET C,7,(IY+nn) FDCBnnF9
SET D,0,(IX+nn) DDCBnnC2
SET D,0,(IY+nn) FDCBnnC2
SET D,1,(IX+nn) DDCBnnCA
SET D,1,(IY+nn) FDCBnnCA
SET D,2,(IX+nn) DDCBnnD2
SET D,2,(IY+nn) FDCBnnD2
SET D,3,(IX+nn) DDCBnnDA
SET D,3,(IY+nn) FDCBnnDA
SET D,4,(IX+nn) DDCBnnE2
SET D,4,(IY+nn) FDCBnnE2
SET D,5,(IX+nn) DDCBnnEA
SET D,5,(IY+nn) FDCBnnEA
SET D,6,(IX+nn) DDCBnnF2
SET D,6,(IY+nn) FDCBnnF2
SET D,7,(IX+nn) DDCBnnFA
SET D,7,(IY+nn) FDCBnnFA
SET E,0,(IX+nn) DDCBnnC3
SET E,0,(IY+nn) FDCBnnC3
SET E,1,(IX+nn) DDCBnnCB
SET E,1,(IY+nn) FDCBnnCB
SET E,2,(IX+nn) DDCBnnD3
SET E,2,(IY+nn) FDCBnnD3
SET E,3,(IX+nn) DDCBnnDB
SET E,3,(IY+nn) FDCBnnDB
SET E,4,(IX+nn) DDCBnnE3
SET E,4,(IY+nn) FDCBnnE3
SET E,5,(IX+nn) DDCBnnEB
SET E,5,(IY+nn) FDCBnnEB
SET E,6,(IX+nn) DDCBnnF3
SET E,6,(IY+nn) FDCBnnF3
SET E,7,(IX+nn) DDCBnnFB
SET E,7,(IY+nn) FDCBnnFB
SET H,0,(IX+nn) DDCBnnC4
SET H,0,(IY+nn) FDCBnnC4
SET H,1,(IX+nn) DDCBnnCC
SET H,1,(IY+nn) FDCBnnCC
SET H,2,(IX+nn) DDCBnnD4
SET H,2,(IY+nn) FDCBnnD4
SET H,3,(IX+nn) DDCBnnDC
SET H,3,(IY+nn) FDCBnnDC
SET H,4,(IX+nn) DDCBnnE4
SET H,4,(IY+nn) FDCBnnE4
SET H,5,(IX+nn) DDCBnnEC
SET H,5,(IY+nn) FDCBnnEC
SET H,6,(IX+nn) DDCBnnF4
SET H,6,(IY+nn) FDCBnnF4
SET H,7,(IX+nn) DDCBnnFC
SET H,7,(IY+nn) FDCBnnFC
SET L,0,(IX+nn) DDCBnnC5
SET L,0,(IY+nn) FDCBnnC5
SET L,1,(IX+nn) DDCBnnCD
SET L,1,(IY+nn) FDCBnnCD
SET L,2,(IX+nn) DDCBnnD5
SET L,2,(IY+nn) FDCBnnD5
SET L,3,(IX+nn) DDCBnnDD
SET L,3,(IY+nn) FDCBnnDD
SET L,4,(IX+nn) DDCBnnE5
SET L,4,(IY+nn) FDCBnnE5
SET L,5,(IX+nn) DDCBnnED
SET L,5,(IY+nn) FDCBnnED
SET L,6,(IX+nn) DDCBnnF5
SET L,6,(IY+nn) FDCBnnF5
SET L,7,(IX+nn) DDCBnnFD
SET L,7,(IY+nn) FDCBnnFD
SLA (HL) CB26nn
SLA (IX+d) DDCBnn26
SLA (IY+d) FDCBnn26
SLA A CB27nn
SLA A,(IX+d) DDCBnn27
SLA A,(IY+d) FDCBnn27
SLA B CB20nn
SLA B,(IX+d) DDCBnn20
SLA B,(IY+d) FDCBnn20
SLA C CB21nn
SLA C,(IX+d) DDCBnn21
SLA C,(IY+d) FDCBnn21
SLA D CB22nn
SLA D,(IX+d) DDCBnn22
SLA D,(IY+d) FDCBnn22
SLA E CB23nn
SLA E,(IX+d) DDCBnn23
SLA E,(IY+d) FDCBnn23
SLA H CB24nn
SLA H,(IX+d) DDCBnn24
SLA H,(IY+d) FDCBnn24
SLA L CB25nn
SLA L,(IX+d) DDCBnn25
SLA L,(IY+d) FDCBnn25
SLL (HL) CB36
SLL (IX+dd) DDCBnn36
SLL (IY+dd) FDCBnn36
SLL A CB37
SLL A,(IX+d) DDCBnn37
SLL A,(IY+d) FDCBnn37
SLL B CB30
SLL B,(IX+d) DDCBnn30
SLL B,(IY+d) FDCBnn30
SLL C CB31
SLL C,(IX+d) DDCBnn31
SLL C,(IY+d) FDCBnn31
SLL D CB32
SLL D,(IX+d) DDCBnn32
SLL D,(IY+d) FDCBnn32
SLL E CB33
SLL E,(IX+d) DDCBnn33
SLL E,(IY+d) FDCBnn33
SLL H CB34
SLL H,(IX+d) DDCBnn34
SLL H,(IY+d) FDCBnn34
SLL L CB35
SLL L,(IX+d) DDCBnn35
SLL L,(IY+d) FDCBnn35
SRA (HL) CB2Enn
SRA (IX+d) DDCBnn2E
SRA (IY+d) FDCBnn2E
SRA A CB2Fnn
SRA A,(IX+d) DDCBnn2F
SRA A,(IY+d) FDCBnn2F
SRA B CB28nn
SRA B,(IX+d) DDCBnn28
SRA B,(IY+d) FDCBnn28
SRA C CB29nn
SRA C,(IX+d) DDCBnn29
SRA C,(IY+d) FDCBnn29
SRA D CB2Ann
SRA D,(IX+d) DDCBnn2A
SRA D,(IY+d) FDCBnn2A
SRA E CB2Bnn
SRA E,(IX+d) DDCBnn2B
SRA E,(IY+d) FDCBnn2B
SRA H CB2Cnn
SRA H,(IX+d) DDCBnn2C
SRA H,(IY+d) FDCBnn2C
SRA L CB2Dnn
SRA L,(IX+d) DDCBnn2D
SRA L,(IY+d) FDCBnn2D
SRL (HL) CB3Enn
SRL (IX+d) DDCBnn3E
SRL (IY+d) FDCBnn3E
SRL A CB3Fnn
SRL A,(IX+d) DDCBnn3F
SRL A,(IY+d) FDCBnn3F
SRL B CB38nn
SRL B,(IX+d) DDCBnn38
SRL B,(IY+d) FDCBnn38
SRL C CB39nn
SRL C,(IX+d) DDCBnn39
SRL C,(IY+d) FDCBnn39
SRL D CB3Ann
SRL D,(IX+d) DDCBnn3A
SRL D,(IY+d) FDCBnn3A
SRL E CB3Bnn
SRL E,(IX+d) DDCBnn3B
SRL E,(IY+d) FDCBnn3B
SRL H CB3Cnn
SRL H,(IX+d) DDCBnn3C
SRL H,(IY+d) FDCBnn3C
SRL L CB3Dnn
SRL L,(IX+d) DDCBnn3D
SRL L,(IY+d) FDCBnn3D
SUB A,(HL) 96

SUB A,(IX+d)	DD96nn	SUB A,L	95	XOR A,(IY+d)	FDAEnn	XOR A,n	EEnn
SUB A,(IY+d)	FD96nn	SUB A,n	D6nn	XOR A,A	AF	*XOR IXh*	*DDAC*
SUB A,A	97	*SUB IXh*	*DD94*	XOR A,B	A8	*XOR IXl*	*DDAD*
SUB A,B	90	*SUB IXl*	*DD95*	XOR A,C	A9	*XOR IYh*	*FDAC*
SUB A,C	91	*SUB IYh*	*FD94*	XOR A,D	AA	*XOR IYl*	*FDAD*
SUB A,D	92	*SUB IYl*	*FD95*	XOR A,E	AB		
SUB A,E	93	XOR A,(HL)	AE	XOR A,H	AC		
SUB A,H	94	XOR A,(IX+d)	DDAEnn	XOR A,L	AD		

6.2 - Instruction List by opcode

NOP	00	DEC SP	3B	HALT	76	OR A,D	B2
LD BC, nn	01nnnn	INC A	3C	LD (HL), A	77	OR A,E	B3
LD (BC), A	02	DEC A	3D	LD A, B	78	OR A,H	B4
INC BC	03	LD A, n	3Enn	LD A, C	79	OR A,L	B5
INC B	04	CCF	3F	LD A, D	7A	OR A,(HL)	B6
DEC B	05	CCF	3F	LD A, E	7B	OR A,A	B7
LD B, n	06nn	LD B, B	40	LD A, H	7C	CP B	B8
RLCA	07	LD B, C	41	LD A, L	7D	CP C	B9
EX AF, AF'	08	LD B, D	42	LD A, (HL)	7E	CP D	BA
ADD HL,BC	09	LD B, E	43	LD A, A	7F	CP E	BB
LD A, (BC)	0A	LD B, H	44	ADD A,B	80	CP H	BC
DEC BC	0B	LD B, L	45	ADD A,C	81	CP L	BD
INC C	0C	LD B, (HL)	46	ADD A,D	82	CP (HL)	BE
DEC C	0D	LD B, A	47	ADD A,E	83	CP A	BF
LD C, n	0Enn	LD C, B	48	ADD A,H	84	RET NZ	C0
RRCA	0F	LD C, C	49	ADD A,L	85	POP BC	C1
DJNZ e	10nn	LD C, D	4A	ADD A,(HL)	86	JP NZ,nn	C2nnnn
LD DE, nn	11nnnn	LD C, E	4B	ADD A,A	87	JP nn	C3nnnn
LD (DE), A	12	LD C, H	4C	ADC A,B	88	CALL NZ,nn	C4nnnn
INC DE	13	LD C, L	4D	ADC A,C	89	PUSH BC	C5
INC D	14	LD C, (HL)	4E	ADC A,D	8A	ADD A,n	C6nn
DEC D	15	LD C, A	4F	ADC A,E	8B	RST 0	C7
LD D, n	16nn	LD D, B	50	ADC A,H	8C	RET Z	C8
RLA	17	LD D, C	51	ADC A,L	8D	RET	C9
JR e	18nn	LD D, D	52	ADC A,(HL)	8E	JP Z,nn	CAnnnn
ADD HL,DE	19	LD D, E	53	ADC A,A	8F	RLC B	CB00nn
LD A, (DE)	1A	LD D, H	54	SUB A,B	90	RLC C	CB01nn
DEC DE	1B	LD D, L	55	SUB A,C	91	RLC D	CB02nn
INC E	1C	LD D, (HL)	56	SUB A,D	92	RLC E	CB03nn
DEC E	1D	LD D, A	57	SUB A,E	93	RLC H	CB04nn
LD E, n	1Enn	LD E, B	58	SUB A,H	94	RLC L	CB05nn
RRA	1F	LD E, C	59	SUB A,L	95	RLC (HL)	CB06nn
JR NZ,e	20nn	LD E, D	5A	SUB A,(HL)	96	RLC A	CB07nn
LD HL, nn	21nnnn	LD E, E	5B	SUB A,A	97	RRC B	CB08nn
LD (nn), HL	22nnnn	LD E, H	5C	SBC A,B	98	RRC C	CB09nn
INC HL	23	LD E, L	5D	SBC A,C	99	RRC D	CB0Ann
INC H	24	LD E, (HL)	5E	SBC A,D	9A	RRC E	CB0Bnn
DEC H	25	LD E, A	5F	SBC A,E	9B	RRC H	CB0Cnn
LD H, n	26nn	LD H, B	60	SBC A,H	9C	RRC L	CB0Dnn
DAA	27	LD H, C	61	SBC A,L	9D	RRC (HL)	CB0Enn
JR Z,e	28nn	LD H, D	62	SBC A,(HL)	9E	RRC A	CB0Fnn
ADD HL,HL	29	LD H, E	63	SBC A,A	9F	RL B	CB10nn
LD HL, (nn)	2Annnn	LD H, H	64	AND A,B	A0	RL C	CB11nn
DEC HL	2B	LD H, L	65	AND A,C	A1	RL D	CB12nn
INC L	2C	LD H, (HL)	66	AND A,D	A2	RL E	CB13nn
DEC L	2D	LD H, A	67	AND A,E	A3	RL H	CB14nn
LD L, n	2Enn	LD L, B	68	AND A,H	A4	RL L	CB15nn
CPL	2F	LD L, C	69	AND A,L	A5	RL (HL)	CB16nn
CPL	2F	LD L, D	6A	AND A,(HL)	A6	RL A	CB17nn
JR NC,e	30nn	LD L, E	6B	AND A,A	A7	RR B	CB18nn
LD SP, nn	31nnnn	LD L, H	6C	XOR A,B	A8	RR C	CB19nn
LD (nn), A	32nnnn	LD L, L	6D	XOR A,C	A9	RR D	CB1Ann
INC SP	33	LD L, (HL)	6E	XOR A,D	AA	RR E	CB1Bnn
INC (HL)	34	LD L, A	6F	XOR A,E	AB	RR H	CB1Cnn
DEC (HL)	35	LD (HL), B	70	XOR A,H	AC	RR L	CB1Dnn
LD (HL), n	36nn	LD (HL), C	71	XOR A,L	AD	RR (HL)	CB1Enn
SCF	37	LD (HL), D	72	XOR A,(HL)	AE	RR A	CB1Fnn
JR C,e	38nn	LD (HL), E	73	XOR A,A	AF	SLA B	CB20nn
ADD HL,SP	39	LD (HL), H	74	OR A,B	B0	SLA C	CB21nn
LD A, (nn)	3Annnn	LD (HL), L	75	OR A,C	B1	SLA D	CB22nn

SLA E	CB23nn	BIT 4,C	CB61nn	RES 3,A	CB9Fnn	SET 3,L	CBDDnn
SLA H	CB24nn	BIT 4,D	CB62nn	RES 4,B	CBA0nn	SET 3,(HL)	CBDEnn
SLA L	CB25nn	BIT 4,E	CB63nn	RES 4,C	CBA1nn	SET 3,A	CBDFnn
SLA (HL)	CB26nn	BIT 4,H	CB64nn	RES 4,D	CBA2nn	SET 4,B	CBE0nn
SLA A	CB27nn	BIT 4,L	CB65nn	RES 4,E	CBA3nn	SET 4,C	CBE1nn
SRA B	CB28nn	BIT 4,(HL)	CB66nn	RES 4,H	CBA4nn	SET 4,D	CBE2nn
SRA C	CB29nn	BIT 4,A	CB67nn	RES 4,L	CBA5nn	SET 4,E	CBE3nn
SRA D	CB2Ann	BIT 5,B	CB68nn	RES 4,(HL)	CBA6nn	SET 4,H	CBE4nn
SRA E	CB2Bnn	BIT 5,C	CB69nn	RES 4,A	CBA7nn	SET 4,L	CBE5nn
SRA H	CB2Cnn	BIT 5,D	CB6Ann	RES 5,B	CBA8nn	SET 4,(HL)	CBE6nn
SRA L	CB2Dnn	BIT 5,E	CB6Bnn	RES 5,C	CBA9nn	SET 4,A	CBE7nn
SRA (HL)	CB2Enn	BIT 5,H	CB6Cnn	RES 5,D	CBAAnn	SET 5,B	CBE8nn
SRA A	CB2Fnn	BIT 5,L	CB6Dnn	RES 5,E	CBABnn	SET 5,C	CBE9nn
SLL B	*CB30*	BIT 5,(HL)	CB6Enn	RES 5,H	CBACnn	SET 5,D	CBEAnn
SLL C	*CB31*	BIT 5,A	CB6Fnn	RES 5,L	CBADnn	SET 5,E	CBEBnn
SLL D	*CB32*	BIT 6,B	CB70nn	RES 5,(HL)	CBAEnn	SET 5,H	CBECnn
SLL E	*CB33*	BIT 6,C	CB71nn	RES 5,A	CBAFnn	SET 5,L	CBEDnn
SLL H	*CB34*	BIT 6,D	CB72nn	RES 6,B	CBB0nn	SET 5,(HL)	CBEEnn
SLL L	*CB35*	BIT 6,E	CB73nn	RES 6,C	CBB1nn	SET 5,A	CBEFnn
SLL (HL)	*CB36*	BIT 6,H	CB74nn	RES 6,D	CBB2nn	SET 6,B	CBF0nn
SLL A	*CB37*	BIT 6,L	CB75nn	RES 6,E	CBB3nn	SET 6,C	CBF1nn
SRL B	CB38nn	BIT 6,(HL)	CB76nn	RES 6,H	CBB4nn	SET 6,D	CBF2nn
SRL C	CB39nn	BIT 6,A	CB77nn	RES 6,L	CBB5nn	SET 6,E	CBF3nn
SRL D	CB3Ann	BIT 7,B	CB78nn	RES 6,(HL)	CBB6nn	SET 6,H	CBF4nn
SRL E	CB3Bnn	BIT 7,C	CB79nn	RES 6,A	CBB7nn	SET 6,L	CBF5nn
SRL H	CB3Cnn	BIT 7,D	CB7Ann	RES 7,B	CBB8nn	SET 6,(HL)	CBF6nn
SRL L	CB3Dnn	BIT 7,E	CB7Bnn	RES 7,C	CBB9nn	SET 6,A	CBF7nn
SRL (HL)	CB3Enn	BIT 7,H	CB7Cnn	RES 7,D	CBBAnn	SET 7,B	CBF8nn
SRL A	CB3Fnn	BIT 7,L	CB7Dnn	RES 7,E	CBBBnn	SET 7,C	CBF9nn
BIT 0,B	CB40nn	BIT 7,(HL)	CB7Enn	RES 7,H	CBBCnn	SET 7,D	CBFAnn
BIT 0,C	CB41nn	BIT 7,A	CB7Fnn	RES 7,L	CBBDnn	SET 7,E	CBFBnn
BIT 0,D	CB42nn	RES 0,B	CB80nn	RES 7,(HL)	CBBEnn	SET 7,H	CBFCnn
BIT 0,E	CB43nn	RES 0,C	CB81nn	RES 7,A	CBBFnn	SET 7,L	CBFDnn
BIT 0,H	CB44nn	RES 0,D	CB82nn	SET 0,B	CBC0nn	SET 7,(HL)	CBFEnn
BIT 0,L	CB45nn	RES 0,E	CB83nn	SET 0,C	CBC1nn	SET 7,A	CBFFnn
BIT 0,(HL)	CB46nn	RES 0,H	CB84nn	SET 0,D	CBC2nn	CALL Z,nn	CCnnnn
BIT 0,A	CB47nn	RES 0,L	CB85nn	SET 0,E	CBC3nn	CALL nn	CDnnnn
BIT 1,B	CB48nn	RES 0,(HL)	CB86nn	SET 0,H	CBC4nn	ADC A,n	CEnn
BIT 1,C	CB49nn	RES 0,A	CB87nn	SET 0,L	CBC5nn	RST 1	CF
BIT 1,D	CB4Ann	RES 1,B	CB88nn	SET 0,(HL)	CBC6nn	RET NC	D0
BIT 1,E	CB4Bnn	RES 1,C	CB89nn	SET 0,A	CBC7nn	POP DE	D1
BIT 1,H	CB4Cnn	RES 1,D	CB8Ann	SET 1,B	CBC8nn	JP NC,nn	D2nnnn
BIT 1,L	CB4Dnn	RES 1,E	CB8Bnn	SET 1,C	CBC9nn	OUT (n),A	D3nn
BIT 1,(HL)	CB4Enn	RES 1,H	CB8Cnn	SET 1,D	CBCAnn	CALL NC,nn	D4nnnn
BIT 1,A	CB4Fnn	RES 1,L	CB8Dnn	SET 1,E	CBCBnn	PUSH DE	D5
BIT 2,B	CB50nn	RES 1,(HL)	CB8Enn	SET 1,H	CBCCnn	SUB A,n	D6nn
BIT 2,C	CB51nn	RES 1,A	CB8Fnn	SET 1,L	CBCDnn	RST 2	D7
BIT 2,D	CB52nn	RES 2,B	CB90nn	SET 1,(HL)	CBCEnn	RET C	D8
BIT 2,E	CB53nn	RES 2,C	CB91nn	SET 1,A	CBCFnn	EXX	D9
BIT 2,H	CB54nn	RES 2,D	CB92nn	SET 2,B	CBD0nn	JP C,nn	DAnnnn
BIT 2,L	CB55nn	RES 2,E	CB93nn	SET 2,C	CBD1nn	IN A,(n)	DBnn
BIT 2,(HL)	CB56nn	RES 2,H	CB94nn	SET 2,D	CBD2nn	CALL C,nn	DCnnnn
BIT 2,A	CB57nn	RES 2,L	CB95nn	SET 2,E	CBD3nn	ADD IX,BC	DD09nn
BIT 3,B	CB58nn	RES 2,(HL)	CB96nn	SET 2,H	CBD4nn	ADD IX,DE	DD19nn
BIT 3,C	CB59nn	RES 2,A	CB97nn	SET 2,L	CBD5nn	LD IX, nn	DD21nnnn
BIT 3,D	CB5Ann	RES 3,B	CB98nn	SET 2,(HL)	CBD6nn	LD (nn), IX	DD22nnnn
BIT 3,E	CB5Bnn	RES 3,C	CB99nn	SET 2,A	CBD7nn	INC IX	DD23nn
BIT 3,H	CB5Cnn	RES 3,D	CB9Ann	SET 3,B	CBD8nn	*INC IXh*	*DD24*
BIT 3,L	CB5Dnn	RES 3,E	CB9Bnn	SET 3,C	CBD9nn	*DEC IXh*	*DD25*
BIT 3,(HL)	CB5Enn	RES 3,H	CB9Cnn	SET 3,D	CBDAnn	*LD IXh,n*	*DD26nn*
BIT 3,A	CB5Fnn	RES 3,L	CB9Dnn	SET 3,E	CBDBnn	ADD IX,IX	DD29nn
BIT 4,B	CB60nn	RES 3,(HL)	CB9Enn	SET 3,H	CBDCnn	LD IX, (nn)	DD2Annnn

DEC IX	DD2Bnn	*XOR IXl*	*DDAD*	*SLL (IX+dd)*	*DDCBnn36*	*BIT 6,(IX+d)*	*DDCBnn74*
INC IXl	*DD2C*	XOR A,(IX+d)	DDAEnn	*SLL A,(IX+d)*	*DDCBnn37*	*BIT 6,(IX+d)*	*DDCBnn75*
DEC IXl	*DD2D*	*OR IXh*	*DDB4*	*SRL B,(IX+d)*	*DDCBnn38*	BIT 6,(IX+d)	DDCBnn76
LD IXl,n	*DD2Enn*	*OR IXl*	*DDB5*	*SRL C,(IX+d)*	*DDCBnn39*	*BIT 6,(IX+d)*	*DDCBnn77*
INC (IX+d)	DD34nn	OR A,(IX+d)	DDB6nn	*SRL D,(IX+d)*	*DDCBnn3A*	*BIT 7,(IX+d)*	*DDCBnn78*
DEC (IX+d)	DD35nn	*CP IXh*	*DDBC*	*SRL E,(IX+d)*	*DDCBnn3B*	*BIT 7,(IX+d)*	*DDCBnn79*
LD (IX+d), n	DD36nnnn	*CP IXl*	*DDBD*	*SRL H,(IX+d)*	*DDCBnn3C*	*BIT 7,(IX+d)*	*DDCBnn7A*
ADD IX,SP	DD39nn	CP (IX+d)	DDBEnn	*SRL L,(IX+d)*	*DDCBnn3D*	*BIT 7,(IX+d)*	*DDCBnn7B*
LD B,IXh	*DD44*	*RLC B,(IX+d)*	*DDCBnn00*	SRL (IX+d)	DDCBnn3E	*BIT 7,(IX+d)*	*DDCBnn7C*
LD B,IXl	*DD45*	*RLC C,(IX+d)*	*DDCBnn01*	*SRL A,(IX+d)*	*DDCBnn3F*	*BIT 7,(IX+d)*	*DDCBnn7D*
LD B, (IX+d)	DD46nn	*RLC D,(IX+d)*	*DDCBnn02*	*BIT 0,(IX+d)*	*DDCBnn40*	BIT 7,(IX+d)	DDCBnn7E
LD C,IXh	*DD4C*	*RLC E,(IX+d)*	*DDCBnn03*	*BIT 0,(IX+d)*	*DDCBnn41*	*BIT 7,(IX+d)*	*DDCBnn7F*
LD C,IXl	*DD4D*	*RLC H,(IX+d)*	*DDCBnn04*	*BIT 0,(IX+d)*	*DDCBnn42*	*RES B,0,(IX+nn)*	*DDCBnn80*
LD C, (IX+d)	DD4Enn	*RLC L,(IX+d)*	*DDCBnn05*	*BIT 0,(IX+d)*	*DDCBnn43*	*RES C,0,(IX+nn)*	*DDCBnn81*
LD D,IXh	*DD54*	RLC (IX+d)	DDCBnn06	*BIT 0,(IX+d)*	*DDCBnn44*	*RES D,0,(IX+nn)*	*DDCBnn82*
LD D,IXl	*DD55*	*RLC A,(IX+d)*	*DDCBnn07*	*BIT 0,(IX+d)*	*DDCBnn45*	*RES E,0,(IX+nn)*	*DDCBnn83*
LD D, (IX+d)	DD56nn	*RRC B,(IX+d)*	*DDCBnn08*	BIT 0,(IX+d)	DDCBnn46	*RES H,0,(IX+nn)*	*DDCBnn84*
LD E,IXh	*DD5C*	*RRC C,(IX+d)*	*DDCBnn09*	*BIT 0,(IX+d)*	*DDCBnn47*	*RES L,0,(IX+nn)*	*DDCBnn85*
LD E,IXl	*DD5D*	*RRC D,(IX+d)*	*DDCBnn0A*	*BIT 1,(IX+d)*	*DDCBnn48*	RES 0,(IX+d)	DDCBnn86
LD E, (IX+d)	DD5Enn	*RRC E,(IX+d)*	*DDCBnn0B*	*BIT 1,(IX+d)*	*DDCBnn49*	*RES A,0,(IX+nn)*	*DDCBnn87*
LD IXh,B	*DD60*	*RRC H,(IX+d)*	*DDCBnn0C*	*BIT 1,(IX+d)*	*DDCBnn4A*	*RES B,1,(IX+nn)*	*DDCBnn88*
LD IXh,C	*DD61*	*RRC L,(IX+d)*	*DDCBnn0D*	*BIT 1,(IX+d)*	*DDCBnn4B*	*RES C,1,(IX+nn)*	*DDCBnn89*
LD IXh,D	*DD62*	RRC (IX+d)	DDCBnn0E	*BIT 1,(IX+d)*	*DDCBnn4C*	*RES D,1,(IX+nn)*	*DDCBnn8A*
LD IXh,E	*DD63*	*RRC A,(IX+d)*	*DDCBnn0F*	*BIT 1,(IX+d)*	*DDCBnn4D*	*RES E,1,(IX+nn)*	*DDCBnn8B*
LD IXh,IHh	*DD64*	*RL B,(IX+d)*	*DDCBnn10*	BIT 1,(IX+d)	DDCBnn4E	*RES H,1,(IX+nn)*	*DDCBnn8C*
LD IXh,IHl	*DD65*	*RL C,(IX+d)*	*DDCBnn11*	*BIT 1,(IX+d)*	*DDCBnn4F*	*RES L,1,(IX+nn)*	*DDCBnn8D*
LD H, (IX+d)	DD66nn	*RL D,(IX+d)*	*DDCBnn12*	*BIT 2,(IX+d)*	*DDCBnn50*	RES 1,(IX+d)	DDCBnn8E
LD IXh,A	*DD67*	*RL E,(IX+d)*	*DDCBnn13*	*BIT 2,(IX+d)*	*DDCBnn51*	*RES A,1,(IX+nn)*	*DDCBnn8F*
LD IXl,B	*DD68*	*RL H,(IX+d)*	*DDCBnn14*	*BIT 2,(IX+d)*	*DDCBnn52*	*RES B,2,(IX+nn)*	*DDCBnn90*
LD IXl,C	*DD69*	*RL L,(IX+d)*	*DDCBnn15*	*BIT 2,(IX+d)*	*DDCBnn53*	*RES C,2,(IX+nn)*	*DDCBnn91*
LD IXl,D	*DD6A*	RL (IX+d)	DDCBnn16	*BIT 2,(IX+d)*	*DDCBnn54*	*RES D,2,(IX+nn)*	*DDCBnn92*
LD IXl,E	*DD6B*	*RL A,(IX+d)*	*DDCBnn17*	*BIT 2,(IX+d)*	*DDCBnn55*	*RES E,2,(IX+nn)*	*DDCBnn93*
LD IXl,IHh	*DD6C*	*RR B,(IX+d)*	*DDCBnn18*	BIT 2,(IX+d)	DDCBnn56	*RES H,2,(IX+nn)*	*DDCBnn94*
LD IXl,IHl	*DD6D*	*RR C,(IX+d)*	*DDCBnn19*	*BIT 2,(IX+d)*	*DDCBnn57*	*RES L,2,(IX+nn)*	*DDCBnn95*
LD L, (IX+d)	DD6Enn	*RR D,(IX+d)*	*DDCBnn1A*	*BIT 3,(IX+d)*	*DDCBnn58*	RES 2,(IX+d)	DDCBnn96
LD IXl,A	*DD6F*	*RR E,(IX+d)*	*DDCBnn1B*	*BIT 3,(IX+d)*	*DDCBnn59*	*RES A,2,(IX+nn)*	*DDCBnn97*
LD (IX+d), B	DD70nn	*RR H,(IX+d)*	*DDCBnn1C*	*BIT 3,(IX+d)*	*DDCBnn5A*	*RES B,3,(IX+nn)*	*DDCBnn98*
LD (IX+d), C	DD71nn	*RR L,(IX+d)*	*DDCBnn1D*	*BIT 3,(IX+d)*	*DDCBnn5B*	*RES C,3,(IX+nn)*	*DDCBnn99*
LD (IX+d), D	DD72nn	RR (IX+d)	DDCBnn1E	*BIT 3,(IX+d)*	*DDCBnn5C*	*RES D,3,(IX+nn)*	*DDCBnn9A*
LD (IX+d), E	DD73nn	*RR A,(IX+d)*	*DDCBnn1F*	*BIT 3,(IX+d)*	*DDCBnn5D*	*RES E,3,(IX+nn)*	*DDCBnn9B*
LD (IX+d), H	DD74nn	*SLA B,(IX+d)*	*DDCBnn20*	BIT 3,(IX+d)	DDCBnn5E	*RES H,3,(IX+nn)*	*DDCBnn9C*
LD (IX+d), L	DD75nn	*SLA C,(IX+d)*	*DDCBnn21*	*BIT 3,(IX+d)*	*DDCBnn5F*	*RES L,3,(IX+nn)*	*DDCBnn9D*
LD (IX+d), A	DD77nn	*SLA D,(IX+d)*	*DDCBnn22*	*BIT 4,(IX+d)*	*DDCBnn60*	RES 3,(IX+d)	DDCBnn9E
LD A,IXh	*DD7C*	*SLA E,(IX+d)*	*DDCBnn23*	*BIT 4,(IX+d)*	*DDCBnn61*	*RES A,3,(IX+nn)*	*DDCBnn9F*
LD A,IXl	*DD7D*	*SLA H,(IX+d)*	*DDCBnn24*	*BIT 4,(IX+d)*	*DDCBnn62*	*RES B,4,(IX+nn)*	*DDCBnnA0*
LD A, (IX+d)	DD7Enn	*SLA L,(IX+d)*	*DDCBnn25*	*BIT 4,(IX+d)*	*DDCBnn63*	*RES C,4,(IX+nn)*	*DDCBnnA1*
ADD A,IXh	*DD84*	SLA (IX+d)	DDCBnn26	*BIT 4,(IX+d)*	*DDCBnn64*	*RES D,4,(IX+nn)*	*DDCBnnA2*
ADD A,IXl	*DD85*	*SLA A,(IX+d)*	*DDCBnn27*	*BIT 4,(IX+d)*	*DDCBnn65*	*RES E,4,(IX+nn)*	*DDCBnnA3*
ADD A,(IX+d)	DD86nn	*SRA B,(IX+d)*	*DDCBnn28*	BIT 4,(IX+d)	DDCBnn66	*RES H,4,(IX+nn)*	*DDCBnnA4*
ADC A,IXh	*DD8C*	*SRA C,(IX+d)*	*DDCBnn29*	*BIT 4,(IX+d)*	*DDCBnn67*	*RES L,4,(IX+nn)*	*DDCBnnA5*
ADC A,IXl	*DD8D*	*SRA D,(IX+d)*	*DDCBnn2A*	*BIT 5,(IX+d)*	*DDCBnn68*	RES 4,(IX+d)	DDCBnnA6
ADC A,(IX+d)	DD8Enn	*SRA E,(IX+d)*	*DDCBnn2B*	*BIT 5,(IX+d)*	*DDCBnn69*	*RES A,4,(IX+nn)*	*DDCBnnA7*
SUB IXh	*DD94*	*SRA H,(IX+d)*	*DDCBnn2C*	*BIT 5,(IX+d)*	*DDCBnn6A*	*RES B,5,(IX+nn)*	*DDCBnnA8*
SUB IXl	*DD95*	*SRA L,(IX+d)*	*DDCBnn2D*	*BIT 5,(IX+d)*	*DDCBnn6B*	*RES C,5,(IX+nn)*	*DDCBnnA9*
SUB A,(IX+d)	DD96nn	SRA (IX+d)	DDCBnn2E	*BIT 5,(IX+d)*	*DDCBnn6C*	*RES D,5,(IX+nn)*	*DDCBnnAA*
SBC A,IXh	*DD9C*	*SRA A,(IX+d)*	*DDCBnn2F*	*BIT 5,(IX+d)*	*DDCBnn6D*	*RES E,5,(IX+nn)*	*DDCBnnAB*
SBC A,IXl	*DD9D*	*SLL B,(IX+d)*	*DDCBnn30*	BIT 5,(IX+d)	DDCBnn6E	*RES H,5,(IX+nn)*	*DDCBnnAC*
SBC A,(IX+d)	DD9Enn	*SLL C,(IX+d)*	*DDCBnn31*	*BIT 5,(IX+d)*	*DDCBnn6F*	*RES L,5,(IX+nn)*	*DDCBnnAD*
AND IXh	*DDA4*	*SLL D,(IX+d)*	*DDCBnn32*	*BIT 6,(IX+d)*	*DDCBnn70*	RES 5,(IX+d)	DDCBnnAE
AND IXl	*DDA5*	*SLL E,(IX+d)*	*DDCBnn33*	*BIT 6,(IX+d)*	*DDCBnn71*	*RES A,5,(IX+nn)*	*DDCBnnAF*
AND A,(IX+d)	DDA6nn	*SLL H,(IX+d)*	*DDCBnn34*	*BIT 6,(IX+d)*	*DDCBnn72*	*RES B,6,(IX+nn)*	*DDCBnnB0*
XOR IXh	*DDAC*	*SLL L,(IX+d)*	*DDCBnn35*	*BIT 6,(IX+d)*	*DDCBnn73*	*RES C,6,(IX+nn)*	*DDCBnnB1*

RES D,6,(IX+nn) *DDCBnnB2*
RES E,6,(IX+nn) *DDCBnnB3*
RES H,6,(IX+nn) *DDCBnnB4*
RES L,6,(IX+nn) *DDCBnnB5*
RES 6,(IX+d) DDCBnnB6
RES A,6,(IX+nn) *DDCBnnB7*
RES B,7,(IX+nn) *DDCBnnB8*
RES C,7,(IX+nn) *DDCBnnB9*
RES D,7,(IX+nn) *DDCBnnBA*
RES E,7,(IX+nn) *DDCBnnBB*
RES H,7,(IX+nn) *DDCBnnBC*
RES L,7,(IX+nn) *DDCBnnBD*
RES 7,(IX+d) DDCBnnBE
RES A,7,(IX+nn) *DDCBnnBF*
SET B,0,(IX+nn) *DDCBnnC0*
SET C,0,(IX+nn) *DDCBnnC1*
SET D,0,(IX+nn) *DDCBnnC2*
SET E,0,(IX+nn) *DDCBnnC3*
SET H,0,(IX+nn) *DDCBnnC4*
SET L,0,(IX+nn) *DDCBnnC5*
SET 0,(IX+d) DDCBnnC6
SET A,0,(IX+nn) *DDCBnnC7*
SET B,1,(IX+nn) *DDCBnnC8*
SET C,1,(IX+nn) *DDCBnnC9*
SET D,1,(IX+nn) *DDCBnnCA*
SET E,1,(IX+nn) *DDCBnnCB*
SET H,1,(IX+nn) *DDCBnnCC*
SET L,1,(IX+nn) *DDCBnnCD*
SET 1,(IX+d) DDCBnnCE
SET A,1,(IX+nn) *DDCBnnCF*
SET B,2,(IX+nn) *DDCBnnD0*
SET C,2,(IX+nn) *DDCBnnD1*
SET D,2,(IX+nn) *DDCBnnD2*
SET E,2,(IX+nn) *DDCBnnD3*
SET H,2,(IX+nn) *DDCBnnD4*
SET L,2,(IX+nn) *DDCBnnD5*
SET 2,(IX+d) DDCBnnD6
SET A,2,(IX+nn) *DDCBnnD7*
SET B,3,(IX+nn) *DDCBnnD8*
SET C,3,(IX+nn) *DDCBnnD9*
SET D,3,(IX+nn) *DDCBnnDA*
SET E,3,(IX+nn) *DDCBnnDB*
SET H,3,(IX+nn) *DDCBnnDC*
SET L,3,(IX+nn) *DDCBnnDD*
SET 3,(IX+d) DDCBnnDE
SET A,3,(IX+nn) *DDCBnnDF*
SET B,4,(IX+nn) *DDCBnnE0*
SET C,4,(IX+nn) *DDCBnnE1*
SET D,4,(IX+nn) *DDCBnnE2*
SET E,4,(IX+nn) *DDCBnnE3*
SET H,4,(IX+nn) *DDCBnnE4*
SET L,4,(IX+nn) *DDCBnnE5*
SET 4,(IX+d) DDCBnnE6
SET A,4,(IX+nn) *DDCBnnE7*
SET B,5,(IX+nn) *DDCBnnE8*
SET C,5,(IX+nn) *DDCBnnE9*
SET D,5,(IX+nn) *DDCBnnEA*
SET E,5,(IX+nn) *DDCBnnEB*
SET H,5,(IX+nn) *DDCBnnEC*
SET L,5,(IX+nn) *DDCBnnED*
SET 5,(IX+d) DDCBnnEE
SET A,5,(IX+nn) *DDCBnnEF*

SET B,6,(IX+nn) *DDCBnnF0*
SET C,6,(IX+nn) *DDCBnnF1*
SET D,6,(IX+nn) *DDCBnnF2*
SET E,6,(IX+nn) *DDCBnnF3*
SET H,6,(IX+nn) *DDCBnnF4*
SET L,6,(IX+nn) *DDCBnnF5*
SET 6,(IX+d) DDCBnnF6
SET A,6,(IX+nn) *DDCBnnF7*
SET B,7,(IX+nn) *DDCBnnF8*
SET C,7,(IX+nn) *DDCBnnF9*
SET D,7,(IX+nn) *DDCBnnFA*
SET E,7,(IX+nn) *DDCBnnFB*
SET H,7,(IX+nn) *DDCBnnFC*
SET L,7,(IX+nn) *DDCBnnFD*
SET 7,(IX+d) DDCBnnFE
SET A,7,(IX+nn) *DDCBnnFF*
POP IX DDE1nn
EX (SP), IX DDE3nn
PUSH IX DDE5nn
JP (IX) DDE9nn
LD SP, IX DDF9nn
SBC A,n DEnn
RST 3 DF
RET PO E0
POP HL E1
JP PO,nn E2nnnn
EX (SP), HL E3
CALL PO,nn E4nnnn
PUSH HL E5
AND A,n E6nn
RST 4 E7
RET PE E8
JP (HL) E9
JP PE,nn EAnnnn
EX DE, HL EB
CALL PE,nn ECnnnn
IN B,(C) ED40nn
OUT (C),B ED41nn
SBC HL,BC ED42nn
LD (nn), BC ED43nnnn
NEG ED44nn
NEG ED44nn
RETN ED45nn
IM0 ED46nn
LD I, A ED47nn
IN C,(C) ED48nn
OUT (C),C ED49nn
ADC HL,BC ED4Ann
LD BC, (nn) ED4Bnnnn
RETI ED4Dnn
LD R, A ED4Fnn
IN D,(C) ED50nn
OUT (C),D ED51nn
SBC HL,DE ED52nn
LD (nn), DE ED53nnnn
IM1 ED56nn
LD A, I ED57nn
IN E,(C) ED58nn
OUT (C),E ED59nn
ADC HL,DE ED5Ann
LD DE, (nn) ED5Bnnnn
IM2 ED5Enn

LD A, R ED5Fnn
IN H,(C) ED60nn
OUT (C),H ED61nn
SBC HL,HL ED62nn
LD (nn), HL ED63nnnn
RRD (HL) ED67nn
IN L,(C) ED68nn
OUT (C),L ED69nn
ADC HL,HL ED6Ann
LD HL, (nn) ED6Bnnnn
RLD (HL) ED6Fnn
IN F,(C) *ED70nn*
OUT (C),F *ED71nn*
SBC HL,SP ED72nn
LD (nn), SP ED73nnnn
OUT (C),A ED79nn
ADC HL,SP ED7Ann
IN A,(C) ED7Bnn
LD SP, (nn) ED7Bnnnn
LDI EDA0nn
CPI EDA1nn
INI EDA2nn
OUTI EDA3nn
LDD EDA8nn
CPD EDA9nn
IND EDAAnn
OUTD EDABnn
LDIR EDB0nn
CPIR EDB1nn
INIR EDB2nn
OUTIR EDB3nn
LDDR EDB8nn
CPDR EDB9nn
INDR EDBAnn
OUTDR EDBBnn
XOR A,n EEnn
RST 5 EF
RET P F0
POP AF F1
JP P,nn F2nnnn
DI F3
CALL P,nn F4nnnn
PUSH AF F5
OR A,n F6nn
RST 6 F7
RET N F8
LD SP, HL F9
JP N,nn FAnnnn
EI FB
CALL N,nn FCnnnn
ADD IY,BC FD09nn
ADD IY,DE FD19nn
LD IY, nn FD21nnnn
LD (nn), IY FD22nnnn
INC IY FD23nn
INC IYh *FD24*
DEC IYh *FD25*
LD IYh,n *FD26nn*
ADD IY,IY FD29nn
LD IY, (nn) FD2Annnn
DEC IY FD2Bnn
INC IYl *FD2C*

DEC IYl *FD2D*
LD IYl,n *FD2Enn*
INC (IY+d) FD34nn
DEC (IY+d) FD35nn
LD (IY+d), n FD36nnnn
ADD IY,SP FD39nn
LD B,IYh *FD44*
LD B,IYl *FD45*
LD B, (IY+d) FD46nn
LD C,IYh *FD4C*
LD C,IYl *FD4D*
LD C, (IY+d) FD4Enn
LD D,IYh *FD54*
LD D,IYl *FD55*
LD D, (IY+d) FD56nn
LD E,IYh *FD5C*
LD E,IYl *FD5D*
LD E, (IY+d) FD5Enn
LD IYh,B *FD60*
LD IYh,C *FD61*
LD IYh,D *FD62*
LD IYh,E *FD63*
LD IYh,IHh *FD64*
LD IYh,IHl *FD65*
LD H, (IY+d) FD66nn
LD IYh,A *FD67*
LD IYl,B *FD68*
LD IYl,C *FD69*
LD IYl,D *FD6A*
LD IYl,E *FD6B*
LD IYl,IHh *FD6C*
LD IYl,IHl *FD6D*
LD L, (IY+d) FD6Enn
LD IYl,A *FD6F*
LD (IY+d), B FD70nn
LD (IY+d), C FD71nn
LD (IY+d), D FD72nn
LD (IY+d), E FD73nn
LD (IY+d), H FD74nn
LD (IY+d), L FD75nn
LD (IY+d), A FD77nn
LD A,IYh *FD7C*
LD A,IYl *FD7D*
LD A, (IY+d) FD7Enn
ADD A,IYh *FD84*
ADD A,IYl *FD85*
ADD A,(IY+d) FD86nn
ADC A,IYh *FD8C*
ADC A,IYl *FD8D*
ADC A,(IY+d) FD8Enn
SUB IYh *FD94*
SUB IYl *FD95*
SUB A,(IY+d) FD96nn
SBC A,IYh *FD9C*
SBC A,IYl *FD9D*
SBC A,(IY+d) FD9Enn
AND IYh *FDA4*
AND IYl *FDA5*
AND A,(IY+d) FDA6nn
XOR IYh *FDAC*
XOR IYl *FDAD*
XOR A,(IY+d) FDAEnn

OR IYh	*FDB4*	*SRL B,(IY+d)*	*FDCBnn38*	BIT 6,(IY+d)	FDCBnn76	*RES H,6,(IY+nn)*	*FDCBnnB4*
OR IYl	*FDB5*	*SRL C,(IY+d)*	*FDCBnn39*	*BIT 6,(IY+d)*	*FDCBnn77*	*RES L,6,(IY+nn)*	*FDCBnnB5*
OR A,(IY+d)	FDB6nn	*SRL D,(IY+d)*	*FDCBnn3A*	*BIT 7,(IY+d)*	*FDCBnn78*	RES 6,(IY+d)	FDCBnnB6
CP IYh	*FDBC*	*SRL E,(IY+d)*	*FDCBnn3B*	*BIT 7,(IY+d)*	*FDCBnn79*	*RES A,6,(IY+nn)*	*FDCBnnB7*
CP IYl	*FDBD*	*SRL H,(IY+d)*	*FDCBnn3C*	*BIT 7,(IY+d)*	*FDCBnn7A*	*RES B,7,(IY+nn)*	*FDCBnnB8*
CP (IY+d)	FDBEnn	*SRL L,(IY+d)*	*FDCBnn3D*	*BIT 7,(IY+d)*	*FDCBnn7B*	*RES C,7,(IY+nn)*	*FDCBnnB9*
RLC B,(IY+d)	*FDCBnn00*	SRL (IY+d)	FDCBnn3E	*BIT 7,(IY+d)*	*FDCBnn7C*	*RES D,7,(IY+nn)*	*FDCBnnBA*
RLC C,(IY+d)	*FDCBnn01*	*SRL A,(IY+d)*	*FDCBnn3F*	*BIT 7,(IY+d)*	*FDCBnn7D*	*RES E,7,(IY+nn)*	*FDCBnnBB*
RLC D,(IY+d)	*FDCBnn02*	*BIT 0,(IY+d)*	*FDCBnn40*	BIT 7,(IY+d)	FDCBnn7E	*RES H,7,(IY+nn)*	*FDCBnnBC*
RLC E,(IY+d)	*FDCBnn03*	*BIT 0,(IY+d)*	*FDCBnn41*	*BIT 7,(IY+d)*	*FDCBnn7F*	*RES L,7,(IY+nn)*	*FDCBnnBD*
RLC H,(IY+d)	*FDCBnn04*	*BIT 0,(IY+d)*	*FDCBnn42*	*RES B,0,(IY+nn)*	*FDCBnn80*	RES 7,(IY+d)	FDCBnnBE
RLC L,(IY+d)	*FDCBnn05*	*BIT 0,(IY+d)*	*FDCBnn43*	*RES C,0,(IY+nn)*	*FDCBnn81*	*RES A,7,(IY+nn)*	*FDCBnnBF*
RLC (IY+d)	FDCBnn06	*BIT 0,(IY+d)*	*FDCBnn44*	*RES D,0,(IY+nn)*	*FDCBnn82*	*SET B,0,(IY+nn)*	*FDCBnnC0*
RLC A,(IY+d)	*FDCBnn07*	*BIT 0,(IY+d)*	*FDCBnn45*	*RES E,0,(IY+nn)*	*FDCBnn83*	*SET C,0,(IY+nn)*	*FDCBnnC1*
RRC B,(IY+d)	*FDCBnn08*	BIT 0,(IY+d)	FDCBnn46	*RES H,0,(IY+nn)*	*FDCBnn84*	*SET D,0,(IY+nn)*	*FDCBnnC2*
RRC C,(IY+d)	*FDCBnn09*	*BIT 0,(IY+d)*	*FDCBnn47*	*RES L,0,(IY+nn)*	*FDCBnn85*	*SET E,0,(IY+nn)*	*FDCBnnC3*
RRC D,(IY+d)	*FDCBnn0A*	*BIT 1,(IY+d)*	*FDCBnn48*	RES 0,(IY+d)	FDCBnn86	*SET H,0,(IY+nn)*	*FDCBnnC4*
RRC E,(IY+d)	*FDCBnn0B*	*BIT 1,(IY+d)*	*FDCBnn49*	*RES A,0,(IY+nn)*	*FDCBnn87*	*SET L,0,(IY+nn)*	*FDCBnnC5*
RRC H,(IY+d)	*FDCBnn0C*	*BIT 1,(IY+d)*	*FDCBnn4A*	*RES B,1,(IY+nn)*	*FDCBnn88*	SET 0,(IY+d)	FDCBnnC6
RRC L,(IY+d)	*FDCBnn0D*	*BIT 1,(IY+d)*	*FDCBnn4B*	*RES C,1,(IY+nn)*	*FDCBnn89*	*SET A,0,(IY+nn)*	*FDCBnnC7*
RRC (IY+d)	FDCBnn0E	*BIT 1,(IY+d)*	*FDCBnn4C*	*RES D,1,(IY+nn)*	*FDCBnn8A*	*SET B,1,(IY+nn)*	*FDCBnnC8*
RRC A,(IY+d)	*FDCBnn0F*	*BIT 1,(IY+d)*	*FDCBnn4D*	*RES E,1,(IY+nn)*	*FDCBnn8B*	*SET C,1,(IY+nn)*	*FDCBnnC9*
RL B,(IY+d)	*FDCBnn10*	BIT 1,(IY+d)	FDCBnn4E	*RES H,1,(IY+nn)*	*FDCBnn8C*	*SET D,1,(IY+nn)*	*FDCBnnCA*
RL C,(IY+d)	*FDCBnn11*	*BIT 1,(IY+d)*	*FDCBnn4F*	*RES L,1,(IY+nn)*	*FDCBnn8D*	*SET E,1,(IY+nn)*	*FDCBnnCB*
RL D,(IY+d)	*FDCBnn12*	*BIT 2,(IY+d)*	*FDCBnn50*	RES 1,(IY+d)	FDCBnn8E	*SET H,1,(IY+nn)*	*FDCBnnCC*
RL E,(IY+d)	*FDCBnn13*	*BIT 2,(IY+d)*	*FDCBnn51*	*RES A,1,(IY+nn)*	*FDCBnn8F*	*SET L,1,(IY+nn)*	*FDCBnnCD*
RL H,(IY+d)	*FDCBnn14*	*BIT 2,(IY+d)*	*FDCBnn52*	*RES B,2,(IY+nn)*	*FDCBnn90*	SET 1,(IY+d)	FDCBnnCE
RL L,(IY+d)	*FDCBnn15*	*BIT 2,(IY+d)*	*FDCBnn53*	*RES C,2,(IY+nn)*	*FDCBnn91*	*SET A,1,(IY+nn)*	*FDCBnnCF*
RL (IY+d)	FDCBnn16	*BIT 2,(IY+d)*	*FDCBnn54*	*RES D,2,(IY+nn)*	*FDCBnn92*	*SET B,2,(IY+nn)*	*FDCBnnD0*
RL A,(IY+d)	*FDCBnn17*	*BIT 2,(IY+d)*	*FDCBnn55*	*RES E,2,(IY+nn)*	*FDCBnn93*	*SET C,2,(IY+nn)*	*FDCBnnD1*
RR B,(IY+d)	*FDCBnn18*	BIT 2,(IY+d)	FDCBnn56	*RES H,2,(IY+nn)*	*FDCBnn94*	*SET D,2,(IY+nn)*	*FDCBnnD2*
RR C,(IY+d)	*FDCBnn19*	*BIT 2,(IY+d)*	*FDCBnn57*	*RES L,2,(IY+nn)*	*FDCBnn95*	*SET E,2,(IY+nn)*	*FDCBnnD3*
RR D,(IY+d)	*FDCBnn1A*	*BIT 3,(IY+d)*	*FDCBnn58*	RES 2,(IY+d)	FDCBnn96	*SET H,2,(IY+nn)*	*FDCBnnD4*
RR E,(IY+d)	*FDCBnn1B*	*BIT 3,(IY+d)*	*FDCBnn59*	*RES A,2,(IY+nn)*	*FDCBnn97*	*SET L,2,(IY+nn)*	*FDCBnnD5*
RR H,(IY+d)	*FDCBnn1C*	*BIT 3,(IY+d)*	*FDCBnn5A*	*RES B,3,(IY+nn)*	*FDCBnn98*	SET 2,(IY+d)	FDCBnnD6
RR L,(IY+d)	*FDCBnn1D*	*BIT 3,(IY+d)*	*FDCBnn5B*	*RES C,3,(IY+nn)*	*FDCBnn99*	*SET A,2,(IY+nn)*	*FDCBnnD7*
RR (IY+d)	FDCBnn1E	*BIT 3,(IY+d)*	*FDCBnn5C*	*RES D,3,(IY+nn)*	*FDCBnn9A*	*SET B,3,(IY+nn)*	*FDCBnnD8*
RR A,(IY+d)	*FDCBnn1F*	*BIT 3,(IY+d)*	*FDCBnn5D*	*RES E,3,(IY+nn)*	*FDCBnn9B*	*SET C,3,(IY+nn)*	*FDCBnnD9*
SLA B,(IY+d)	*FDCBnn20*	BIT 3,(IY+d)	FDCBnn5E	*RES H,3,(IY+nn)*	*FDCBnn9C*	*SET D,3,(IY+nn)*	*FDCBnnDA*
SLA C,(IY+d)	*FDCBnn21*	*BIT 3,(IY+d)*	*FDCBnn5F*	*RES L,3,(IY+nn)*	*FDCBnn9D*	*SET E,3,(IY+nn)*	*FDCBnnDB*
SLA D,(IY+d)	*FDCBnn22*	*BIT 4,(IY+d)*	*FDCBnn60*	RES 3,(IY+d)	FDCBnn9E	*SET H,3,(IY+nn)*	*FDCBnnDC*
SLA E,(IY+d)	*FDCBnn23*	*BIT 4,(IY+d)*	*FDCBnn61*	*RES A,3,(IY+nn)*	*FDCBnn9F*	*SET L,3,(IY+nn)*	*FDCBnnDD*
SLA H,(IY+d)	*FDCBnn24*	*BIT 4,(IY+d)*	*FDCBnn62*	*RES B,4,(IY+nn)*	*FDCBnnA0*	SET 3,(IY+d)	FDCBnnDE
SLA L,(IY+d)	*FDCBnn25*	*BIT 4,(IY+d)*	*FDCBnn63*	*RES C,4,(IY+nn)*	*FDCBnnA1*	*SET A,3,(IY+nn)*	*FDCBnnDF*
SLA (IY+d)	FDCBnn26	*BIT 4,(IY+d)*	*FDCBnn64*	*RES D,4,(IY+nn)*	*FDCBnnA2*	*SET B,4,(IY+nn)*	*FDCBnnE0*
SLA A,(IY+d)	*FDCBnn27*	*BIT 4,(IY+d)*	*FDCBnn65*	*RES E,4,(IY+nn)*	*FDCBnnA3*	*SET C,4,(IY+nn)*	*FDCBnnE1*
SRA B,(IY+d)	*FDCBnn28*	BIT 4,(IY+d)	FDCBnn66	*RES H,4,(IY+nn)*	*FDCBnnA4*	*SET D,4,(IY+nn)*	*FDCBnnE2*
SRA C,(IY+d)	*FDCBnn29*	*BIT 4,(IY+d)*	*FDCBnn67*	*RES L,4,(IY+nn)*	*FDCBnnA5*	*SET E,4,(IY+nn)*	*FDCBnnE3*
SRA D,(IY+d)	*FDCBnn2A*	*BIT 5,(IY+d)*	*FDCBnn68*	RES 4,(IY+d)	FDCBnnA6	*SET H,4,(IY+nn)*	*FDCBnnE4*
SRA E,(IY+d)	*FDCBnn2B*	*BIT 5,(IY+d)*	*FDCBnn69*	*RES A,4,(IY+nn)*	*FDCBnnA7*	*SET L,4,(IY+nn)*	*FDCBnnE5*
SRA H,(IY+d)	*FDCBnn2C*	*BIT 5,(IY+d)*	*FDCBnn6A*	*RES B,5,(IY+nn)*	*FDCBnnA8*	SET 4,(IY+d)	FDCBnnE6
SRA L,(IY+d)	*FDCBnn2D*	*BIT 5,(IY+d)*	*FDCBnn6B*	*RES C,5,(IY+nn)*	*FDCBnnA9*	*SET A,4,(IY+nn)*	*FDCBnnE7*
SRA (IY+d)	FDCBnn2E	*BIT 5,(IY+d)*	*FDCBnn6C*	*RES D,5,(IY+nn)*	*FDCBnnAA*	*SET B,5,(IY+nn)*	*FDCBnnE8*
SRA A,(IY+d)	*FDCBnn2F*	*BIT 5,(IY+d)*	*FDCBnn6D*	*RES E,5,(IY+nn)*	*FDCBnnAB*	*SET C,5,(IY+nn)*	*FDCBnnE9*
SLL B,(IY+d)	*FDCBnn30*	BIT 5,(IY+d)	FDCBnn6E	*RES H,5,(IY+nn)*	*FDCBnnAC*	*SET D,5,(IY+nn)*	*FDCBnnEA*
SLL C,(IY+d)	*FDCBnn31*	*BIT 5,(IY+d)*	*FDCBnn6F*	*RES L,5,(IY+nn)*	*FDCBnnAD*	*SET E,5,(IY+nn)*	*FDCBnnEB*
SLL D,(IY+d)	*FDCBnn32*	*BIT 6,(IY+d)*	*FDCBnn70*	RES 5,(IY+d)	FDCBnnAE	*SET H,5,(IY+nn)*	*FDCBnnEC*
SLL E,(IY+d)	*FDCBnn33*	*BIT 6,(IY+d)*	*FDCBnn71*	*RES A,5,(IY+nn)*	*FDCBnnAF*	*SET L,5,(IY+nn)*	*FDCBnnED*
SLL H,(IY+d)	*FDCBnn34*	*BIT 6,(IY+d)*	*FDCBnn72*	*RES B,6,(IY+nn)*	*FDCBnnB0*	SET 5,(IY+d)	FDCBnnEE
SLL L,(IY+d)	*FDCBnn35*	*BIT 6,(IY+d)*	*FDCBnn73*	*RES C,6,(IY+nn)*	*FDCBnnB1*	*SET A,5,(IY+nn)*	*FDCBnnEF*
SLL (IY+dd)	*FDCBnn36*	*BIT 6,(IY+d)*	*FDCBnn74*	*RES D,6,(IY+nn)*	*FDCBnnB2*	*SET B,6,(IY+nn)*	*FDCBnnF0*
SLL A,(IY+d)	*FDCBnn37*	*BIT 6,(IY+d)*	*FDCBnn75*	*RES E,6,(IY+nn)*	*FDCBnnB3*	*SET C,6,(IY+nn)*	*FDCBnnF1*

SET D,6,(IY+nn)	*FDCBnnF2*	*SET B,7,(IY+nn)*	*FDCBnnF8*	SET 7,(IY+d)	FDCBnnFE	LD SP, IY	FDF9nn
SET E,6,(IY+nn)	*FDCBnnF3*	*SET C,7,(IY+nn)*	*FDCBnnF9*	*SET A,7,(IY+nn)*	*FDCBnnFF*	CP n	FEnn
SET H,6,(IY+nn)	*FDCBnnF4*	*SET D,7,(IY+nn)*	*FDCBnnFA*	POP IY	FDE1nn	RST 7	FF
SET L,6,(IY+nn)	*FDCBnnF5*	*SET E,7,(IY+nn)*	*FDCBnnFB*	EX (SP), IY	FDE3nn		
SET 6,(IY+d)	FDCBnnF6	*SET H,7,(IY+nn)*	*FDCBnnFC*	PUSH IY	FDE5nn		
SET A,6,(IY+nn)	*FDCBnnF7*	*SET L,7,(IY+nn)*	*FDCBnnFD*	JP (IY)	FDE9nn		

6.3 - Opcode Matrix

Instructions shown in an Opcode Matrix

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP 1 4 00	LD BC, nn 3 10 01nnnn	LD (BC), A 1 7 02	INC BC 1 6 03	INC B 1 4 04	DEC B 1 4 05	LD B, n 2 7 06nn	RLCA 1 4 07	EX AF, AF' 1 4 08	ADD HL,BC 1 11 09	LD A, (BC) 1 7 0A	DEC BC 1 6 0B	INC C 1 4 0C	DEC C 1 4 0D	LD C, n 2 7 0Enn	RRCA 1 4 0F
1	DJNZ e 2 13 10nn	LD DE, nn 3 10 11nnnn	LD (DE), A 1 7 12	INC DE 1 6 13	INC D 1 4 14	DEC D 1 4 15	LD D, n 2 7 16nn	RLA 1 4 17	JR e 2 12 18nn	ADD HL,DE 1 11 19	LD A, (DE) 1 7 1A	DEC DE 1 6 1B	INC E 1 4 1C	DEC E 1 4 1D	LD E, n 2 7 1Enn	RRA 1 4 1F
2	JR NZ,e 2 12 20nn	LD HL, nn 3 10 21nnnn	LD (nn), HL 3 16 22nnnn	INC HL 1 6 23	INC H 1 4 24	DEC H 1 4 25	LD H, n 2 7 26nn	DAA 1 4 27	JR Z,e 2 12 28nn	ADD HL,HL 1 11 29	LD HL, (nn) 3 16 2Annnn	DEC HL 1 6 2B	INC L 1 4 2C	DEC L 1 4 2D	LD L, n 2 7 2Enn	CPL 1 4 2F
3	JR NC,e 2 12 30nn	LD SP, nn 3 10 31nnnn	LD (nn), A 3 13 32nnnn	INC SP 1 6 33	INC (HL) 1 11 34	DEC (HL) 1 11 35	LD (HL), n 2 10 36nn	SCF 1 4 37	JR C,e 2 12 38nn	ADD HL,SP 1 11 39	LD A, (nn) 3 13 3Annnn	DEC SP 1 6 3B	INC A 1 4 3C	DEC A 1 4 3D	LD A, n 2 7 3Enn	CCF 1 4 3F
4	LD B, B 1 4 40	LD B, C 1 4 41	LD B, D 1 4 42	LD B, E 1 4 43	LD B, H 1 4 44	LD B, L 1 4 45	LD B, (HL) 1 7 46	LD B, A 1 4 47	LD C, B 1 4 48	LD C, C 1 4 49	LD C, D 1 4 4A	LD C, E 1 4 4B	LD C, H 1 4 4C	LD C, L 1 4 4D	LD C, (HL) 1 7 4E	LD C, A 1 4 4F
5	LD D, B 1 4 50	LD D, C 1 4 51	LD D, D 1 4 52	LD D, E 1 4 53	LD D, H 1 4 54	LD D, L 1 4 55	LD D, (HL) 1 7 56	LD D, A 1 4 57	LD E, B 1 4 58	LD E, C 1 4 59	LD E, D 1 4 5A	LD E, E 1 4 5B	LD E, H 1 4 5C	LD E, L 1 4 5D	LD E, (HL) 1 7 5E	LD E, A 1 4 5F
6	LD H, B 1 4 60	LD H, C 1 4 61	LD H, D 1 4 62	LD H, E 1 4 63	LD H, H 1 4 64	LD H, L 1 4 65	LD H, (HL) 1 7 66	LD H, A 1 4 67	LD L, B 1 4 68	LD L, C 1 4 69	LD L, D 1 4 6A	LD L, E 1 4 6B	LD L, H 1 4 6C	LD L, L 1 4 6D	LD L, (HL) 1 7 6E	LD L, A 1 4 6F
7	LD (HL), B 1 7 70	LD (HL), C 1 7 71	LD (HL), D 1 7 72	LD (HL), E 1 7 73	LD (HL), H 1 7 74	LD (HL), L 1 7 75	HALT 1 4 76	LD (HL), A 1 7 77	LD A, B 1 4 78	LD A, C 1 4 79	LD A, D 1 4 7A	LD A, E 1 4 7B	LD A, H 1 4 7C	LD A, L 1 4 7D	LD A, (HL) 1 7 7E	LD A, A 1 4 7F
8	ADD A,B 1 4 80	ADD A,C 1 4 81	ADD A,D 1 4 82	ADD A,E 1 4 83	ADD A,H 1 4 84	ADD A,L 1 4 85	ADD A,(HL) 1 7 86	ADD A,A 1 4 87	ADC A,B 1 4 88	ADC A,C 1 4 89	ADC A,D 1 4 8A	ADC A,E 1 4 8B	ADC A,H 1 4 8C	ADC A,L 1 4 8D	ADC A,(HL) 1 7 8E	ADC A,A 1 4 8F
9	SUB A,B 1 4 90	SUB A,C 1 4 91	SUB A,D 1 4 92	SUB A,E 1 4 93	SUB A,H 1 4 94	SUB A,L 1 4 95	SUB A,(HL) 1 7 96	SUB A,A 1 4 97	SBC A,B 1 4 98	SBC A,C 1 4 99	SBC A,D 1 4 9A	SBC A,E 1 4 9B	SBC A,H 1 4 9C	SBC A,L 1 4 9D	SBC A,(HL) 1 7 9E	SBC A,A 1 4 9F
A	AND A,B 1 4 A0	AND A,C 1 4 A1	AND A,D 1 4 A2	AND A,E 1 4 A3	AND A,H 1 4 A4	AND A,L 1 4 A5	AND A,(HL) 1 7 A6	AND A,A 1 4 A7	XOR A,B 1 4 A8	XOR A,C 1 4 A9	XOR A,D 1 4 AA	XOR A,E 1 4 AB	XOR A,H 1 4 AC	XOR A,L 1 4 AD	XOR A,(HL) 1 7 AE	XOR A,A 1 4 AF
B	OR A,B 1 4 B0	OR A,C 1 4 B1	OR A,D 1 4 B2	OR A,E 1 4 B3	OR A,H 1 4 B4	OR A,L 1 4 B5	OR A,(HL) 1 7 B6	OR A,A 1 4 B7	CP B 1 4 B8	CP C 1 4 B9	CP D 1 4 BA	CP E 1 4 BB	CP H 1 4 BC	CP L 1 4 BD	CP (HL) 1 7 BE	CP A 1 4 BF
C	RET NZ 1 11 C0	POP BC 1 10 C1	JP NZ,nn 3 10 C2nnnn	JP nn 3 10 C3nnnn	CALL NZ,nn 3 17 C4nnnn	PUSH BC 1 11 C5	ADD A,n 2 7 C6nn	RST 0 1 11 C7	RET Z 1 11 C8	RET 1 10 C9	JP Z,nn 3 10 CAnnnn	Instruction Prefix CB	CALL Z,nn 3 17 CCnnnn	CALL nn 3 17 CDnnnn	ADC A,n 2 7 CEnn	RST 1 1 11 CF
D	RET NC 1 11 D0	POP DE 1 10 D1	JP NC,nn 3 10 D2nnnn	OUT (n),A 2 11 D3nn	CALL NC,nn 3 17 D4nnnn	PUSH DE 1 11 D5	SUB A,n 2 7 D6nn	RST 2 1 11 D7	RET C 1 11 D8	EXX 1 4 D9	JP C,nn 3 10 DAnnnn	IN A,(n) 2 11 DBnn	CALL C,nn 3 17 DCnnnn	Instruction Prefix DD	SBC A,n 2 7 DEnn	RST 3 1 11 DF
E	RET PO 1 11 E0	POP HL 1 10 E1	JP PO,nn 3 10 E2nnnn	EX (SP), HL 1 19 E3	CALL PO,nn 3 17 E4nnnn	PUSH HL 1 11 E5	AND A,n 2 7 E6nn	RST 4 1 11 E7	RET PE 1 11 E8	JP (HL) 1 4 E9	JP PE,nn 3 10 EAnnnn	EX DE, HL 1 4 EB	CALL PE,nn 3 17 ECnnnn	Instruction Prefix ED	XOR A,n 2 7 EEnn	RST 5 1 11 EF
F	RET P 1 11 F0	POP AF 1 10 F1	JP P,nn 3 10 F2nnnn	DI 1 4 F3	CALL P,nn 3 17 F4nnnn	PUSH AF 1 11 F5	OR A,n 2 7 F6nn	RST 6 1 11 F7	RET N 1 11 F8	LD SP, HL 1 6 F9	JP N,nn 3 10 FAnnnn	EI 1 4 FB	CALL N,nn 3 17 FCnnnn	Instruction Prefix FD	CP n 2 7 FEnn	RST 7 1 11 FF

Opcode Matrix Legend

Size bytes / Instruction / Opcode hex / Cycle count	Register	Memory	Implicit	Flow	Interrupt	Special	Extension	Undefined	Undocumented

Opcodes with prefix 0xCB

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	RLC B 2 8 CB00nn	RLC C 2 8 CB01nn	RLC D 2 8 CB02nn	RLC E 2 8 CB03nn	RLC H 2 8 CB04nn	RLC L 2 8 CB05nn	RLC (HL) 2 8 CB06nn	RLC A 2 8 CB07nn	RRC B 2 8 CB08nn	RRC C 2 8 CB09nn	RRC D 2 8 CB0Ann	RRC E 2 8 CB0Bnn	RRC H 2 8 CB0Cnn	RRC L 2 8 CB0Dnn	RRC (HL) 2 15 CB0Enn	RRC A 2 8 CB0Fnn
1	RL B 2 8 CB10nn	RL C 2 8 CB11nn	RL D 2 8 CB12nn	RL E 2 8 CB13nn	RL H 2 8 CB14nn	RL L 2 8 CB15nn	RL (HL) 2 15 CB16nn	RL A 2 8 CB17nn	RR B 2 8 CB18nn	RR C 2 8 CB19nn	RR D 2 8 CB1Ann	RR E 2 8 CB1Bnn	RR H 2 8 CB1Cnn	RR L 2 8 CB1Dnn	RR (HL) 2 15 CB1Enn	RR A 2 8 CB1Fnn
2	SLA B 2 8 CB20nn	SLA C 2 8 CB21nn	SLA D 2 8 CB22nn	SLA E 2 8 CB23nn	SLA H 2 8 CB24nn	SLA L 2 8 CB25nn	SLA (HL) 2 15 CB26nn	SLA A 2 8 CB27nn	SRA B 2 8 CB28nn	SRA C 2 8 CB29nn	SRA D 2 8 CB2Ann	SRA E 2 8 CB2Bnn	SRA H 2 8 CB2Cnn	SRA L 2 8 CB2Dnn	SRA (HL) 2 15 CB2Enn	SRA A 2 8 CB2Fnn
3	SLL B CB30	SLL C CB31	SLL D CB32	SLL E CB33	SLL H CB34	SLL L CB35	SLL (HL) CB36	SLL A CB37	SRL B 2 8 CB38nn	SRL C 2 8 CB39nn	SRL D 2 8 CB3Ann	SRL E 2 8 CB3Bnn	SRL H 2 8 CB3Cnn	SRL L 2 8 CB3Dnn	SRL (HL) 2 15 CB3Enn	SRL A 2 8 CB3Fnn
4	BIT 0,B 2 8 CB40nn	BIT 0,C 2 8 CB41nn	BIT 0,D 2 8 CB42nn	BIT 0,E 2 8 CB43nn	BIT 0,H 2 8 CB44nn	BIT 0,L 2 8 CB45nn	BIT 0,(HL) 2 12 CB46nn	BIT 0,A 2 8 CB47nn	BIT 1,B 2 8 CB48nn	BIT 1,C 2 8 CB49nn	BIT 1,D 2 8 CB4Ann	BIT 1,E 2 8 CB4Bnn	BIT 1,H 2 8 CB4Cnn	BIT 1,L 2 8 CB4Dnn	BIT 1,(HL) 2 12 CB4Enn	BIT 1,A 2 8 CB4Fnn
5	BIT 2,B 2 8 CB50nn	BIT 2,C 2 8 CB51nn	BIT 2,D 2 8 CB52nn	BIT 2,E 2 8 CB53nn	BIT 2,H 2 8 CB54nn	BIT 2,L 2 8 CB55nn	BIT 2,(HL) 2 12 CB56nn	BIT 2,A 2 8 CB57nn	BIT 3,B 2 8 CB58nn	BIT 3,C 2 8 CB59nn	BIT 3,D 2 8 CB5Ann	BIT 3,E 2 8 CB5Bnn	BIT 3,H 2 8 CB5Cnn	BIT 3,L 2 8 CB5Dnn	BIT 3,(HL) 2 12 CB5Enn	BIT 3,A 2 8 CB5Fnn
6	BIT 4,B 2 8 CB60nn	BIT 4,C 2 8 CB61nn	BIT 4,D 2 8 CB62nn	BIT 4,E 2 8 CB63nn	BIT 4,H 2 8 CB64nn	BIT 4,L 2 8 CB65nn	BIT 4,(HL) 2 12 CB66nn	BIT 4,A 2 8 CB67nn	BIT 5,B 2 8 CB68nn	BIT 5,C 2 8 CB69nn	BIT 5,D 2 8 CB6Ann	BIT 5,E 2 8 CB6Bnn	BIT 5,H 2 8 CB6Cnn	BIT 5,L 2 8 CB6Dnn	BIT 5,(HL) 2 12 CB6Enn	BIT 5,A 2 8 CB6Fnn
7	BIT 6,B 2 8 CB70nn	BIT 6,C 2 8 CB71nn	BIT 6,D 2 8 CB72nn	BIT 6,E 2 8 CB73nn	BIT 6,H 2 8 CB74nn	BIT 6,L 2 8 CB75nn	BIT 6,(HL) 2 12 CB76nn	BIT 6,A 2 8 CB77nn	BIT 7,B 2 8 CB78nn	BIT 7,C 2 8 CB79nn	BIT 7,D 2 8 CB7Ann	BIT 7,E 2 8 CB7Bnn	BIT 7,H 2 8 CB7Cnn	BIT 7,L 2 8 CB7Dnn	BIT 7,(HL) 2 12 CB7Enn	BIT 7,A 2 8 CB7Fnn
8	RES 0,B 2 8 CB80nn	RES 0,C 2 8 CB81nn	RES 0,D 2 8 CB82nn	RES 0,E 2 8 CB83nn	RES 0,H 2 8 CB84nn	RES 0,L 2 8 CB85nn	RES 0,(HL) 2 15 CB86nn	RES 0,A 2 8 CB87nn	RES 1,B 2 8 CB88nn	RES 1,C 2 8 CB89nn	RES 1,D 2 8 CB8Ann	RES 1,E 2 8 CB8Bnn	RES 1,H 2 8 CB8Cnn	RES 1,L 2 8 CB8Dnn	RES 1,(HL) 2 15 CB8Enn	RES 1,A 2 8 CB8Fnn
9	RES 2,B 2 8 CB90nn	RES 2,C 2 8 CB91nn	RES 2,D 2 8 CB92nn	RES 2,E 2 8 CB93nn	RES 2,H 2 8 CB94nn	RES 2,L 2 8 CB95nn	RES 2,(HL) 2 15 CB96nn	RES 2,A 2 8 CB97nn	RES 3,B 2 8 CB98nn	RES 3,C 2 8 CB99nn	RES 3,D 2 8 CB9Ann	RES 3,E 2 8 CB9Bnn	RES 3,H 2 8 CB9Cnn	RES 3,L 2 8 CB9Dnn	RES 3,(HL) 2 15 CB9Enn	RES 3,A 2 8 CB9Fnn
A	RES 4,B 2 8 CBA0nn	RES 4,C 2 8 CBA1nn	RES 4,D 2 8 CBA2nn	RES 4,E 2 8 CBA3nn	RES 4,H 2 8 CBA4nn	RES 4,L 2 8 CBA5nn	RES 4,(HL) 2 15 CBA6nn	RES 4,A 2 8 CBA7nn	RES 5,B 2 8 CBA8nn	RES 5,C 2 8 CBA9nn	RES 5,D 2 8 CBAAnn	RES 5,E 2 8 CBABnn	RES 5,H 2 8 CBACnn	RES 5,L 2 8 CBADnn	RES 5,(HL) 2 15 CBAEnn	RES 5,A 2 8 CBAFnn
B	RES 6,B 2 8 CBB0nn	RES 6,C 2 8 CBB1nn	RES 6,D 2 8 CBB2nn	RES 6,E 2 8 CBB3nn	RES 6,H 2 8 CBB4nn	RES 6,L 2 8 CBB5nn	RES 6,(HL) 2 15 CBB6nn	RES 6,A 2 8 CBB7nn	RES 7,B 2 8 CBB8nn	RES 7,C 2 8 CBB9nn	RES 7,D 2 8 CBBAnn	RES 7,E 2 8 CBBBnn	RES 7,H 2 8 CBBCnn	RES 7,L 2 8 CBBDnn	RES 7,(HL) 2 15 CBBEnn	RES 7,A 2 8 CBBFnn
C	SET 0,B 2 8 CBC0nn	SET 0,C 2 8 CBC1nn	SET 0,D 2 8 CBC2nn	SET 0,E 2 8 CBC3nn	SET 0,H 2 8 CBC4nn	SET 0,L 2 8 CBC5nn	SET 0,(HL) 2 15 CBC6nn	SET 0,A 2 8 CBC7nn	SET 1,B 2 8 CBC8nn	SET 1,C 2 8 CBC9nn	SET 1,D 2 8 CBCAnn	SET 1,E 2 8 CBCBnn	SET 1,H 2 8 CBCCnn	SET 1,L 2 8 CBCDnn	SET 1,(HL) 2 15 CBCEnn	SET 1,A 2 8 CBCFnn
D	SET 2,B 2 8 CBD0nn	SET 2,C 2 8 CBD1nn	SET 2,D 2 8 CBD2nn	SET 2,E 2 8 CBD3nn	SET 2,H 2 8 CBD4nn	SET 2,L 2 8 CBD5nn	SET 2,(HL) 2 15 CBD6nn	SET 2,A 2 8 CBD7nn	SET 3,B 2 8 CBD8nn	SET 3,C 2 8 CBD9nn	SET 3,D 2 8 CBDAnn	SET 3,E 2 8 CBDBnn	SET 3,H 2 8 CBDCnn	SET 3,L 2 8 CBDDnn	SET 3,(HL) 2 15 CBDEnn	SET 3,A 2 8 CBDFnn
E	SET 4,B 2 8 CBE0nn	SET 4,C 2 8 CBE1nn	SET 4,D 2 8 CBE2nn	SET 4,E 2 8 CBE3nn	SET 4,H 2 8 CBE4nn	SET 4,L 2 8 CBE5nn	SET 4,(HL) 2 15 CBE6nn	SET 4,A 2 8 CBE7nn	SET 5,B 2 8 CBE8nn	SET 5,C 2 8 CBE9nn	SET 5,D 2 8 CBEAnn	SET 5,E 2 8 CBEBnn	SET 5,H 2 8 CBECnn	SET 5,L 2 8 CBEDnn	SET 5,(HL) 2 15 CBEEnn	SET 5,A 2 8 CBEFnn
F	SET 6,B 2 8 CBF0nn	SET 6,C 2 8 CBF1nn	SET 6,D 2 8 CBF2nn	SET 6,E 2 8 CBF3nn	SET 6,H 2 8 CBF4nn	SET 6,L 2 8 CBF5nn	SET 6,(HL) 2 15 CBF6nn	SET 6,A 2 8 CBF7nn	SET 7,B 2 8 CBF8nn	SET 7,C 2 8 CBF9nn	SET 7,D 2 8 CBFAnn	SET 7,E 2 8 CBFBnn	SET 7,H 2 8 CBFCnn	SET 7,L 2 8 CBFDnn	SET 7,(HL) 2 15 CBFEnn	SET 7,A 2 8 CBFFnn

Opcodes with prefix 0xDD

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0										ADD IX,BC 2 15 DD09nn						
1										ADD IX,DE 2 15 DD19nn						
2		LD IX, nn 4 14 DD21nnnn	LD (nn), IX 4 20 DD22nnnn	INC IX 2 10 DD23nn	INC IXh DD24	DEC IXh DD25	LD IXh,n DD26nn			ADD IX,IX 2 15 DD29nn	LD IX, (nn) 4 20 DD2Annnn	DEC IX 2 10 DD2Bnn	INC IXl DD2C	DEC IXl DD2D	LD IXl,n DD2Enn	
3					INC (IX+d) 3 23 DD34nn	DEC (IX+d) 3 23 DD35nn	LD (IX+d), n 4 19 DD36nnnn			ADD IX,SP 2 15 DD39nn						
4					LD B,IXh DD44	LD B,IXl DD45	LD B, (IX+d) 3 19 DD46nn						LD C,IXh DD4C	LD C,IXl DD4D	LD C, (IX+d) 3 19 DD4Enn	
5					LD D,IXh DD54	LD D,IXl DD55	LD D, (IX+d) 3 19 DD56nn						LD E,IXh DD5C	LD E,IXl DD5D	LD E, (IX+d) 3 19 DD5Enn	
6	LD IXh,B DD60	LD IXh,C DD61	LD IXh,D DD62	LD IXh,E DD63	LD IXh,IXh DD64	LD IXh,IXl DD65	LD H, (IX+d) 3 19 DD66nn	LD IXh,A DD67	LD IXl,B DD68	LD IXl,C DD69	LD IXl,D DD6A	LD IXl,E DD6B	LD IXl,IXh DD6C	LD IXl,IXl DD6D	LD L, (IX+d) 3 19 DD6Enn	LD IXl,A DD6F
7	LD (IX+d), B 3 19 DD70nn	LD (IX+d), C 3 19 DD71nn	LD (IX+d), D 3 19 DD72nn	LD (IX+d), E 3 19 DD73nn	LD (IX+d), H 3 19 DD74nn	LD (IX+d), L 3 19 DD75nn		LD (IX+d), A 3 19 DD77nn					LD A,IXh DD7C	LD A,IXl DD7D	LD A, (IX+d) 3 19 DD7Enn	
8					ADD A,IXh DD84	ADD A,IXl DD85	ADD A,(IX+d) 3 19 DD86nn						ADC A,IXh DD8C	ADC A,IXl DD8D	ADC A,(IX+d) 3 19 DD8Enn	
9					SUB IXh DD94	SUB IXl DD95	SUB A,(IX+d) 3 19 DD96nn						SBC A,IXh DD9C	SBC A,IXl DD9D	SBC A,(IX+d) 1 19 DD9Enn	
A					AND IXh DDA4	AND IXl DDA5	AND A,(IX+d) 3 19 DDA6nn						XOR IXh DDAC	XOR IXl DDAD	XOR A,(IX+d) 3 19 DDAEnn	
B					OR IXh DDB4	OR IXl DDB5	OR A,(IX+d) 3 19 DDB6nn						CP IXh DDBC	CP IXl DDBD	CP (IX+d) 3 19 DDBEnn	
C												Instruction Prefix DDCB				
D																
E		POP IX 2 14 DDE1nn		EX (SP), IX 2 23 DDE3nn		PUSH IX 2 15 DDE5nn				JP (IX) 2 8 DDE9nn						
F										LD SP, IX 2 6 DDF9nn						

Opcodes with prefix 0xDDCB

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	RLC B,(IX+d) DDCBnn00	RLC C,(IX+d) DDCBnn01	RLC D,(IX+d) DDCBnn02	RLC E,(IX+d) DDCBnn03	RLC H,(IX+d) DDCBnn04	RLC L,(IX+d) DDCBnn05	RLC (IX+d) 4 23 DDCBnn06	RLC A,(IX+d) DDCBnn07	RRC B,(IX+d) DDCBnn08	RRC C,(IX+d) DDCBnn09	RRC D,(IX+d) DDCBnn0A	RRC E,(IX+d) DDCBnn0B	RRC H,(IX+d) DDCBnn0C	RRC L,(IX+d) DDCBnn0D	RRC (IX+d) 4 23 DDCBnn0E	RRC A,(IX+d) DDCBnn0F
1	RL B,(IX+d) DDCBnn10	RL C,(IX+d) DDCBnn11	RL D,(IX+d) DDCBnn12	RL E,(IX+d) DDCBnn13	RL H,(IX+d) DDCBnn14	RL L,(IX+d) DDCBnn15	RL (IX+d) 4 23 DDCBnn16	RL A,(IX+d) DDCBnn17	RR B,(IX+d) DDCBnn18	RR C,(IX+d) DDCBnn19	RR D,(IX+d) DDCBnn1A	RR E,(IX+d) DDCBnn1B	RR H,(IX+d) DDCBnn1C	RR L,(IX+d) DDCBnn1D	RR (IX+d) 4 23 DDCBnn1E	RR A,(IX+d) DDCBnn1F
2	SLA B,(IX+d) DDCBnn20	SLA C,(IX+d) DDCBnn21	SLA D,(IX+d) DDCBnn22	SLA E,(IX+d) DDCBnn23	SLA H,(IX+d) DDCBnn24	SLA L,(IX+d) DDCBnn25	SLA (IX+d) 4 23 DDCBnn26	SLA A,(IX+d) DDCBnn27	SRA B,(IX+d) DDCBnn28	SRA C,(IX+d) DDCBnn29	SRA D,(IX+d) DDCBnn2A	SRA E,(IX+d) DDCBnn2B	SRA H,(IX+d) DDCBnn2C	SRA L,(IX+d) DDCBnn2D	SRA (IX+d) 4 23 DDCBnn2E	SRA A,(IX+d) DDCBnn2F
3	SLL B,(IX+d) DDCBnn30	SLL C,(IX+d) DDCBnn31	SLL D,(IX+d) DDCBnn32	SLL E,(IX+d) DDCBnn33	SLL H,(IX+d) DDCBnn34	SLL L,(IX+d) DDCBnn35	SLL (IX+dd) DDCBnn36	SLL A,(IX+d) DDCBnn37	SRL B,(IX+d) DDCBnn38	SRL C,(IX+d) DDCBnn39	SRL D,(IX+d) DDCBnn3A	SRL E,(IX+d) DDCBnn3B	SRL H,(IX+d) DDCBnn3C	SRL L,(IX+d) DDCBnn3D	SRL (IX+d) 4 23 DDCBnn3E	SRL A,(IX+d) DDCBnn3F
4	BIT 0,(IX+d) DDCBnn40	BIT 0,(IX+d) DDCBnn41	BIT 0,(IX+d) DDCBnn42	BIT 0,(IX+d) DDCBnn43	BIT 0,(IX+d) DDCBnn44	BIT 0,(IX+d) DDCBnn45	BIT 0,(IX+d) 4 20 DDCBnn46	BIT 0,(IX+d) DDCBnn47	BIT 1,(IX+d) DDCBnn48	BIT 1,(IX+d) DDCBnn49	BIT 1,(IX+d) DDCBnn4A	BIT 1,(IX+d) DDCBnn4B	BIT 1,(IX+d) DDCBnn4C	BIT 1,(IX+d) DDCBnn4D	BIT 1,(IX+d) 4 20 DDCBnn4E	BIT 1,(IX+d) DDCBnn4F
5	BIT 2,(IX+d) DDCBnn50	BIT 2,(IX+d) DDCBnn51	BIT 2,(IX+d) DDCBnn52	BIT 2,(IX+d) DDCBnn53	BIT 2,(IX+d) DDCBnn54	BIT 2,(IX+d) DDCBnn55	BIT 2,(IX+d) 4 20 DDCBnn56	BIT 2,(IX+d) DDCBnn57	BIT 3,(IX+d) DDCBnn58	BIT 3,(IX+d) DDCBnn59	BIT 3,(IX+d) DDCBnn5A	BIT 3,(IX+d) DDCBnn5B	BIT 3,(IX+d) DDCBnn5C	BIT 3,(IX+d) DDCBnn5D	BIT 3,(IX+d) 4 20 DDCBnn5E	BIT 3,(IX+d) DDCBnn5F
6	BIT 4,(IX+d) DDCBnn60	BIT 4,(IX+d) DDCBnn61	BIT 4,(IX+d) DDCBnn62	BIT 4,(IX+d) DDCBnn63	BIT 4,(IX+d) DDCBnn64	BIT 4,(IX+d) DDCBnn65	BIT 4,(IX+d) 4 20 DDCBnn66	BIT 4,(IX+d) DDCBnn67	BIT 5,(IX+d) DDCBnn68	BIT 5,(IX+d) DDCBnn69	BIT 5,(IX+d) DDCBnn6A	BIT 5,(IX+d) DDCBnn6B	BIT 5,(IX+d) DDCBnn6C	BIT 5,(IX+d) DDCBnn6D	BIT 5,(IX+d) 4 20 DDCBnn6E	BIT 5,(IX+d) DDCBnn6F
7	BIT 6,(IX+d) DDCBnn70	BIT 6,(IX+d) DDCBnn71	BIT 6,(IX+d) DDCBnn72	BIT 6,(IX+d) DDCBnn73	BIT 6,(IX+d) DDCBnn74	BIT 6,(IX+d) DDCBnn75	BIT 6,(IX+d) 4 20 DDCBnn76	BIT 6,(IX+d) DDCBnn77	BIT 7,(IX+d) DDCBnn78	BIT 7,(IX+d) DDCBnn79	BIT 7,(IX+d) DDCBnn7A	BIT 7,(IX+d) DDCBnn7B	BIT 7,(IX+d) DDCBnn7C	BIT 7,(IX+d) DDCBnn7D	BIT 7,(IX+d) 4 20 DDCBnn7E	BIT 7,(IX+d) DDCBnn7F
8	RES B,0, (IX+nn) DDCBnn80	RES C,0, (IX+nn) DDCBnn81	RES D,0, (IX+nn) DDCBnn82	RES E,0, (IX+nn) DDCBnn83	RES H,0, (IX+nn) DDCBnn84	RES L,0, (IX+nn) DDCBnn85	RES 0,(IX+d) 4 23 DDCBnn86	RES A,0, (IX+nn) DDCBnn87	RES B,1, (IX+nn) DDCBnn88	RES C,1, (IX+nn) DDCBnn89	RES D,1, (IX+nn) DDCBnn8A	RES E,1, (IX+nn) DDCBnn8B	RES H,1, (IX+nn) DDCBnn8C	RES L,1, (IX+nn) DDCBnn8D	RES 1,(IX+d) 4 23 DDCBnn8E	RES A,1, (IX+nn) DDCBnn8F
9	RES B,2, (IX+nn) DDCBnn90	RES C,2, (IX+nn) DDCBnn91	RES D,2, (IX+nn) DDCBnn92	RES E,2, (IX+nn) DDCBnn93	RES H,2, (IX+nn) DDCBnn94	RES L,2, (IX+nn) DDCBnn95	RES 2,(IX+d) 4 23 DDCBnn96	RES A,2, (IX+nn) DDCBnn97	RES B,3, (IX+nn) DDCBnn98	RES C,3, (IX+nn) DDCBnn99	RES D,3, (IX+nn) DDCBnn9A	RES E,3, (IX+nn) DDCBnn9B	RES H,3, (IX+nn) DDCBnn9C	RES L,3, (IX+nn) DDCBnn9D	RES 3,(IX+d) 4 23 DDCBnn9E	RES A,3, (IX+nn) DDCBnn9F
A	RES B,4, (IX+nn) DDCBnnA0	RES C,4, (IX+nn) DDCBnnA1	RES D,4, (IX+nn) DDCBnnA2	RES E,4, (IX+nn) DDCBnnA3	RES H,4, (IX+nn) DDCBnnA4	RES L,4, (IX+nn) DDCBnnA5	RES 4,(IX+d) 4 23 DDCBnnA6	RES A,4, (IX+nn) DDCBnnA7	RES B,5, (IX+nn) DDCBnnA8	RES C,5, (IX+nn) DDCBnnA9	RES D,5, (IX+nn) DDCBnnAA	RES E,5, (IX+nn) DDCBnnAB	RES H,5, (IX+nn) DDCBnnAC	RES L,5, (IX+nn) DDCBnnAD	RES 5,(IX+d) 4 23 DDCBnnAE	RES A,5, (IX+nn) DDCBnnAF
B	RES B,6, (IX+nn) DDCBnnB0	RES C,6, (IX+nn) DDCBnnB1	RES D,6, (IX+nn) DDCBnnB2	RES E,6, (IX+nn) DDCBnnB3	RES H,6, (IX+nn) DDCBnnB4	RES L,6, (IX+nn) DDCBnnB5	RES 6,(IX+d) 4 23 DDCBnnB6	RES A,6, (IX+nn) DDCBnnB7	RES B,7, (IX+nn) DDCBnnB8	RES C,7, (IX+nn) DDCBnnB9	RES D,7, (IX+nn) DDCBnnBA	RES E,7, (IX+nn) DDCBnnBB	RES H,7, (IX+nn) DDCBnnBC	RES L,7, (IX+nn) DDCBnnBD	RES 7,(IX+d) 4 23 DDCBnnBE	RES A,7, (IX+nn) DDCBnnBF
C	SET B,0, (IX+nn) DDCBnnC0	SET C,0, (IX+nn) DDCBnnC1	SET D,0, (IX+nn) DDCBnnC2	SET E,0, (IX+nn) DDCBnnC3	SET H,0, (IX+nn) DDCBnnC4	SET L,0, (IX+nn) DDCBnnC5	SET 0,(IX+d) 4 23 DDCBnnC6	SET A,0, (IX+nn) DDCBnnC7	SET B,1, (IX+nn) DDCBnnC8	SET C,1, (IX+nn) DDCBnnC9	SET D,1, (IX+nn) DDCBnnCA	SET E,1, (IX+nn) DDCBnnCB	SET H,1, (IX+nn) DDCBnnCC	SET L,1, (IX+nn) DDCBnnCD	SET 1,(IX+d) 4 23 DDCBnnCE	SET A,1, (IX+nn) DDCBnnCF
D	SET B,2, (IX+nn) DDCBnnD0	SET C,2, (IX+nn) DDCBnnD1	SET D,2, (IX+nn) DDCBnnD2	SET E,2, (IX+nn) DDCBnnD3	SET H,2, (IX+nn) DDCBnnD4	SET L,2, (IX+nn) DDCBnnD5	SET 2,(IX+d) 4 23 DDCBnnD6	SET A,2, (IX+nn) DDCBnnD7	SET B,3, (IX+nn) DDCBnnD8	SET C,3, (IX+nn) DDCBnnD9	SET D,3, (IX+nn) DDCBnnDA	SET E,3, (IX+nn) DDCBnnDB	SET H,3, (IX+nn) DDCBnnDC	SET L,3, (IX+nn) DDCBnnDD	SET 3,(IX+d) 4 23 DDCBnnDE	SET A,3, (IX+nn) DDCBnnDF
E	SET B,4, (IX+nn) DDCBnnE0	SET C,4, (IX+nn) DDCBnnE1	SET D,4, (IX+nn) DDCBnnE2	SET E,4, (IX+nn) DDCBnnE3	SET H,4, (IX+nn) DDCBnnE4	SET L,4, (IX+nn) DDCBnnE5	SET 4,(IX+d) 4 23 DDCBnnE6	SET A,4, (IX+nn) DDCBnnE7	SET B,5, (IX+nn) DDCBnnE8	SET C,5, (IX+nn) DDCBnnE9	SET D,5, (IX+nn) DDCBnnEA	SET E,5, (IX+nn) DDCBnnEB	SET H,5, (IX+nn) DDCBnnEC	SET L,5, (IX+nn) DDCBnnED	SET 5,(IX+d) 4 23 DDCBnnEE	SET A,5, (IX+nn) DDCBnnEF
F	SET B,6, (IX+nn) DDCBnnF0	SET C,6, (IX+nn) DDCBnnF1	SET D,6, (IX+nn) DDCBnnF2	SET E,6, (IX+nn) DDCBnnF3	SET H,6, (IX+nn) DDCBnnF4	SET L,6, (IX+nn) DDCBnnF5	SET 6,(IX+d) 4 23 DDCBnnF6	SET A,6, (IX+nn) DDCBnnF7	SET B,7, (IX+nn) DDCBnnF8	SET C,7, (IX+nn) DDCBnnF9	SET D,7, (IX+nn) DDCBnnFA	SET E,7, (IX+nn) DDCBnnFB	SET H,7, (IX+nn) DDCBnnFC	SET L,7, (IX+nn) DDCBnnFD	SET 7,(IX+d) 4 23 DDCBnnFE	SET A,7, (IX+nn) DDCBnnFF

Opcodes with prefix 0xED

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2																
3																
4	IN B,(C) 2 12 ED40nn	OUT (C),B 2 12 ED41nn	SBC HL,BC 2 15 ED42nn	LD (nn), BC 4 20 ED43nnnn	NEG 2 4 ED44nn	RETN 2 14 ED45nn	IM0 2 8 ED46nn	LD I, A 2 4 ED47nn	IN C,(C) 2 12 ED48nn	OUT (C),C 2 12 ED49nn	ADC HL,BC 2 15 ED4Ann	LD BC, (nn) 4 20 ED4Bnnnn		RETI 2 14 ED4Dnn		LD R, A 2 4 ED4Fnn
5	IN D,(C) 2 12 ED50nn	OUT (C),D 2 12 ED51nn	SBC HL,DE 2 15 ED52nn	LD (nn), DE 4 20 ED53nnnn			IM1 2 8 ED56nn	LD A, I 2 9 ED57nn	IN E,(C) 2 12 ED58nn	OUT (C),E 2 12 ED59nn	ADC HL,DE 2 15 ED5Ann	LD DE, (nn) 4 20 ED5Bnnnn			IM2 2 8 ED5Enn	LD A, R 2 9 ED5Fnn
6	IN H,(C) 2 12 ED60nn	OUT (C),H 2 12 ED61nn	SBC HL,HL 2 15 ED62nn	LD (nn), HL 4 20 ED63nnnn				RRD (HL) 2 18 ED67nn	IN L,(C) 2 12 ED68nn	OUT (C),L 2 12 ED69nn	ADC HL,HL 2 15 ED6Ann	LD HL, (nn) 4 20 ED6Bnnnn				RLD (HL) 2 18 ED6Fnn
7	IN F,(C) 2 12 ED70nn	OUT (C),F 2 12 ED71nn	SBC HL,SP 2 15 ED72nn	LD (nn), SP 4 20 ED73nnnn						OUT (C),A 2 12 ED79nn	ADC HL,SP 2 15 ED7Ann	LD SP, (nn) 4 20 ED7Bnnnn				
8																
9																
A	LDI 2 16 EDA0nn	CPI 2 16 EDA1nn	INI 2 16 EDA2nn	OUTI 2 16 EDA3nn					LDD 2 16 EDA8nn	CPD 2 16 EDA9nn	IND 2 16 EDAAnn	OUTD 2 16 EDABnn				
B	LDIR 2 21 EDB0nn	CPIR 2 21 EDB1nn	INIR 2 21 EDB2nn	OUTIR 2 21 EDB3nn					LDDR 2 21 EDB8nn	CPDR 2 21 EDB9nn	INDR 2 21 EDBAnn	OUTDR 2 21 EDBBnn				
C																
D																
E																
F																

Opcodes with prefix 0xFD

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0										ADD IY,BC 2 15 FD09nn						
1										ADD IY,DE 2 15 FD19nn						
2		LD IY, nn 4 14 FD21nnnn	LD (nn), IY 4 20 FD22nnnn	INC IY 2 10 FD23nn	INC IYh FD24	DEC IYh FD25	LD IYh,n FD26nn			ADD IY,IY 2 15 FD29nn	LD IY, (nn) 4 20 FD2Annnn	DEC IY 2 10 FD2Bnn	INC IYl FD2C	DEC IYl FD2D	LD IYl,n FD2Enn	
3					INC (IY+d) 3 23 FD34nn	DEC (IY+d) 3 23 FD35nn	LD (IY+d), n 4 19 FD36nnnn			ADD IY,SP 2 15 FD39nn						
4					LD B,IYh FD44	LD B,IYl FD45	LD B, (IY+d) 3 19 FD46nn						LD C,IYh FD4C	LD C,IYl FD4D	LD C, (IY+d) 3 19 FD4Enn	
5					LD D,IYh FD54	LD D,IYl FD55	LD D, (IY+d) 3 19 FD56nn						LD E,IYh FD5C	LD E,IYl FD5D	LD E, (IY+d) 3 19 FD5Enn	
6	LD IYh,B FD60	LD IYh,C FD61	LD IYh,D FD62	LD IYh,E FD63	LD IYh,IHh FD64	LD IYh,IHl FD65	LD H, (IY+d) 3 19 FD66nn	LD IYh,A FD67	LD IYl,B FD68	LD IYl,C FD69	LD IYl,D FD6A	LD IYl,E FD6B	LD IYl,IHh FD6C	LD IYl,IHl FD6D	LD L, (IY+d) 3 19 FD6Enn	LD IYl,A FD6F
7	LD (IY+d), B 3 19 FD70nn	LD (IY+d), C 3 19 FD71nn	LD (IY+d), D 3 19 FD72nn	LD (IY+d), E 3 19 FD73nn	LD (IY+d), H 3 19 FD74nn	LD (IY+d), L 3 19 FD75nn		LD (IY+d), A 3 19 FD77nn					LD A,IYh FD7C	LD A,IYl FD7D	LD A, (IY+d) 3 19 FD7Enn	
8					ADD A,IYh FD84	ADD A,IYl FD85	ADD A,(IY+d) 3 19 FD86nn						ADC A,IYh FD8C	ADC A,IYl FD8D	ADC A,(IY+d) 3 19 FD8Enn	
9					SUB IYh FD94	SUB IYl FD95	SUB A,(IY+d) 3 19 FD96nn						SBC A,IYh FD9C	SBC A,IYl FD9D	SBC A,(IY+d) 1 19 FD9Enn	
A					AND IYh FDA4	AND IYl FDA5	AND A,(IY+d) 3 19 FDA6nn						XOR IYh FDAC	XOR IYl FDAD	XOR A,(IY+d) 3 19 FDAEnn	
B					OR IYh FDB4	OR IYl FDB5	OR A,(IY+d) 3 19 FDB6nn						CP IYh FDBC	CP IYl FDBD	CP (IY+d) 3 19 FDBEnn	
C												Instruction Prefix FDCB				
D																
E		POP IY 2 14 FDE1nn		EX (SP), IY 2 23 FDE3nn		PUSH IY 2 15 FDE5nn				JP (IY) 2 8 FDE9nn						
F										LD SP, IY 2 6 FDF9nn						

Opcodes with prefix 0xFDCB

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	RLC B,(IY+d) FDCBnn00	RLC C,(IY+d) FDCBnn01	RLC D,(IY+d) FDCBnn02	RLC E,(IY+d) FDCBnn03	RLC H,(IY+d) FDCBnn04	RLC L,(IY+d) FDCBnn05	RLC (IY+d) 4 23 FDCBnn06	RLC A,(IY+d) FDCBnn07	RRC B,(IY+d) FDCBnn08	RRC C,(IY+d) FDCBnn09	RRC D,(IY+d) FDCBnn0A	RRC E,(IY+d) FDCBnn0B	RRC H,(IY+d) FDCBnn0C	RRC L,(IY+d) FDCBnn0D	RRC (IY+d) 4 23 FDCBnn0E	RRC A,(IY+d) FDCBnn0F
1	RL B,(IY+d) FDCBnn10	RL C,(IY+d) FDCBnn11	RL D,(IY+d) FDCBnn12	RL E,(IY+d) FDCBnn13	RL H,(IY+d) FDCBnn14	RL L,(IY+d) FDCBnn15	RL (IY+d) 4 23 FDCBnn16	RL A,(IY+d) FDCBnn17	RR B,(IY+d) FDCBnn18	RR C,(IY+d) FDCBnn19	RR D,(IY+d) FDCBnn1A	RR E,(IY+d) FDCBnn1B	RR H,(IY+d) FDCBnn1C	RR L,(IY+d) FDCBnn1D	RR (IY+d) 4 23 FDCBnn1E	RR A,(IY+d) FDCBnn1F
2	SLA B,(IY+d) FDCBnn20	SLA C,(IY+d) FDCBnn21	SLA D,(IY+d) FDCBnn22	SLA E,(IY+d) FDCBnn23	SLA H,(IY+d) FDCBnn24	SLA L,(IY+d) FDCBnn25	SLA (IY+d) 4 23 FDCBnn26	SLA A,(IY+d) FDCBnn27	SRA B,(IY+d) FDCBnn28	SRA C,(IY+d) FDCBnn29	SRA D,(IY+d) FDCBnn2A	SRA E,(IY+d) FDCBnn2B	SRA H,(IY+d) FDCBnn2C	SRA L,(IY+d) FDCBnn2D	SRA (IY+d) 4 23 FDCBnn2E	SRA A,(IY+d) FDCBnn2F
3	SLL B,(IY+d) FDCBnn30	SLL C,(IY+d) FDCBnn31	SLL D,(IY+d) FDCBnn32	SLL E,(IY+d) FDCBnn33	SLL H,(IY+d) FDCBnn34	SLL L,(IY+d) FDCBnn35	SLL (IY+dd) FDCBnn36	SLL A,(IY+d) FDCBnn37	SRL B,(IY+d) FDCBnn38	SRL C,(IY+d) FDCBnn39	SRL D,(IY+d) FDCBnn3A	SRL E,(IY+d) FDCBnn3B	SRL H,(IY+d) FDCBnn3C	SRL L,(IY+d) FDCBnn3D	SRL (IY+d) 4 23 FDCBnn3E	SRL A,(IY+d) FDCBnn3F
4	BIT 0,(IY+d) FDCBnn40	BIT 0,(IY+d) FDCBnn41	BIT 0,(IY+d) FDCBnn42	BIT 0,(IY+d) FDCBnn43	BIT 0,(IY+d) FDCBnn44	BIT 0,(IY+d) FDCBnn45	BIT 0,(IY+d) 4 20 FDCBnn46	BIT 0,(IY+d) FDCBnn47	BIT 1,(IY+d) FDCBnn48	BIT 1,(IY+d) FDCBnn49	BIT 1,(IY+d) FDCBnn4A	BIT 1,(IY+d) FDCBnn4B	BIT 1,(IY+d) FDCBnn4C	BIT 1,(IY+d) FDCBnn4D	BIT 1,(IY+d) 4 20 FDCBnn4E	BIT 1,(IY+d) FDCBnn4F
5	BIT 2,(IY+d) FDCBnn50	BIT 2,(IY+d) FDCBnn51	BIT 2,(IY+d) FDCBnn52	BIT 2,(IY+d) FDCBnn53	BIT 2,(IY+d) FDCBnn54	BIT 2,(IY+d) FDCBnn55	BIT 2,(IY+d) 4 20 FDCBnn56	BIT 2,(IY+d) FDCBnn57	BIT 3,(IY+d) FDCBnn58	BIT 3,(IY+d) FDCBnn59	BIT 3,(IY+d) FDCBnn5A	BIT 3,(IY+d) FDCBnn5B	BIT 3,(IY+d) FDCBnn5C	BIT 3,(IY+d) FDCBnn5D	BIT 3,(IY+d) 4 20 FDCBnn5E	BIT 3,(IY+d) FDCBnn5F
6	BIT 4,(IY+d) FDCBnn60	BIT 4,(IY+d) FDCBnn61	BIT 4,(IY+d) FDCBnn62	BIT 4,(IY+d) FDCBnn63	BIT 4,(IY+d) FDCBnn64	BIT 4,(IY+d) FDCBnn65	BIT 4,(IY+d) 4 20 FDCBnn66	BIT 4,(IY+d) FDCBnn67	BIT 5,(IY+d) FDCBnn68	BIT 5,(IY+d) FDCBnn69	BIT 5,(IY+d) FDCBnn6A	BIT 5,(IY+d) FDCBnn6B	BIT 5,(IY+d) FDCBnn6C	BIT 5,(IY+d) FDCBnn6D	BIT 5,(IY+d) 4 20 FDCBnn6E	BIT 5,(IY+d) FDCBnn6F
7	BIT 6,(IY+d) FDCBnn70	BIT 6,(IY+d) FDCBnn71	BIT 6,(IY+d) FDCBnn72	BIT 6,(IY+d) FDCBnn73	BIT 6,(IY+d) FDCBnn74	BIT 6,(IY+d) FDCBnn75	BIT 6,(IY+d) 4 20 FDCBnn76	BIT 6,(IY+d) FDCBnn77	BIT 7,(IY+d) FDCBnn78	BIT 7,(IY+d) FDCBnn79	BIT 7,(IY+d) FDCBnn7A	BIT 7,(IY+d) FDCBnn7B	BIT 7,(IY+d) FDCBnn7C	BIT 7,(IY+d) FDCBnn7D	BIT 7,(IY+d) 4 20 FDCBnn7E	BIT 7,(IY+d) FDCBnn7F
8	RES B,0, (IY+nn) FDCBnn80	RES C,0, (IY+nn) FDCBnn81	RES D,0, (IY+nn) FDCBnn82	RES E,0, (IY+nn) FDCBnn83	RES H,0, (IY+nn) FDCBnn84	RES L,0, (IY+nn) FDCBnn85	RES 0,(IY+d) 4 23 FDCBnn86	RES A,0, (IY+nn) FDCBnn87	RES B,1, (IY+nn) FDCBnn88	RES C,1, (IY+nn) FDCBnn89	RES D,1, (IY+nn) FDCBnn8A	RES E,1, (IY+nn) FDCBnn8B	RES H,1, (IY+nn) FDCBnn8C	RES L,1, (IY+nn) FDCBnn8D	RES 1,(IY+d) 4 23 FDCBnn8E	RES A,1, (IY+nn) FDCBnn8F
9	RES B,2, (IY+nn) FDCBnn90	RES C,2, (IY+nn) FDCBnn91	RES D,2, (IY+nn) FDCBnn92	RES E,2, (IY+nn) FDCBnn93	RES H,2, (IY+nn) FDCBnn94	RES L,2, (IY+nn) FDCBnn95	RES 2,(IY+d) 4 23 FDCBnn96	RES A,2, (IY+nn) FDCBnn97	RES B,3, (IY+nn) FDCBnn98	RES C,3, (IY+nn) FDCBnn99	RES D,3, (IY+nn) FDCBnn9A	RES E,3, (IY+nn) FDCBnn9B	RES H,3, (IY+nn) FDCBnn9C	RES L,3, (IY+nn) FDCBnn9D	RES 3,(IY+d) 4 23 FDCBnn9E	RES A,3, (IY+nn) FDCBnn9F
A	RES B,4, (IY+nn) FDCBnnA0	RES C,4, (IY+nn) FDCBnnA1	RES D,4, (IY+nn) FDCBnnA2	RES E,4, (IY+nn) FDCBnnA3	RES H,4, (IY+nn) FDCBnnA4	RES L,4, (IY+nn) FDCBnnA5	RES 4,(IY+d) 4 23 FDCBnnA6	RES A,4, (IY+nn) FDCBnnA7	RES B,5, (IY+nn) FDCBnnA8	RES C,5, (IY+nn) FDCBnnA9	RES D,5, (IY+nn) FDCBnnAA	RES E,5, (IY+nn) FDCBnnAB	RES H,5, (IY+nn) FDCBnnAC	RES L,5, (IY+nn) FDCBnnAD	RES 5,(IY+d) 4 23 FDCBnnAE	RES A,5, (IY+nn) FDCBnnAF
B	RES B,6, (IY+nn) FDCBnnB0	RES C,6, (IY+nn) FDCBnnB1	RES D,6, (IY+nn) FDCBnnB2	RES E,6, (IY+nn) FDCBnnB3	RES H,6, (IY+nn) FDCBnnB4	RES L,6, (IY+nn) FDCBnnB5	RES 6,(IY+d) 4 23 FDCBnnB6	RES A,6, (IY+nn) FDCBnnB7	RES B,7, (IY+nn) FDCBnnB8	RES C,7, (IY+nn) FDCBnnB9	RES D,7, (IY+nn) FDCBnnBA	RES E,7, (IY+nn) FDCBnnBB	RES H,7, (IY+nn) FDCBnnBC	RES L,7, (IY+nn) FDCBnnBD	RES 7,(IY+d) 4 23 FDCBnnBE	RES A,7, (IY+nn) FDCBnnBF
C	SET B,0, (IY+nn) FDCBnnC0	SET C,0, (IY+nn) FDCBnnC1	SET D,0, (IY+nn) FDCBnnC2	SET E,0, (IY+nn) FDCBnnC3	SET H,0, (IY+nn) FDCBnnC4	SET L,0, (IY+nn) FDCBnnC5	SET 0,(IY+d) 4 23 FDCBnnC6	SET A,0, (IY+nn) FDCBnnC7	SET B,1, (IY+nn) FDCBnnC8	SET C,1, (IY+nn) FDCBnnC9	SET D,1, (IY+nn) FDCBnnCA	SET E,1, (IY+nn) FDCBnnCB	SET H,1, (IY+nn) FDCBnnCC	SET L,1, (IY+nn) FDCBnnCD	SET 1,(IY+d) 4 23 FDCBnnCE	SET A,1, (IY+nn) FDCBnnCF
D	SET B,2, (IY+nn) FDCBnnD0	SET C,2, (IY+nn) FDCBnnD1	SET D,2, (IY+nn) FDCBnnD2	SET E,2, (IY+nn) FDCBnnD3	SET H,2, (IY+nn) FDCBnnD4	SET L,2, (IY+nn) FDCBnnD5	SET 2,(IY+d) 4 23 FDCBnnD6	SET A,2, (IY+nn) FDCBnnD7	SET B,3, (IY+nn) FDCBnnD8	SET C,3, (IY+nn) FDCBnnD9	SET D,3, (IY+nn) FDCBnnDA	SET E,3, (IY+nn) FDCBnnDB	SET H,3, (IY+nn) FDCBnnDC	SET L,3, (IY+nn) FDCBnnDD	SET 3,(IY+d) 4 23 FDCBnnDE	SET A,3, (IY+nn) FDCBnnDF
E	SET B,4, (IY+nn) FDCBnnE0	SET C,4, (IY+nn) FDCBnnE1	SET D,4, (IY+nn) FDCBnnE2	SET E,4, (IY+nn) FDCBnnE3	SET H,4, (IY+nn) FDCBnnE4	SET L,4, (IY+nn) FDCBnnE5	SET 4,(IY+d) 4 23 FDCBnnE6	SET A,4, (IY+nn) FDCBnnE7	SET B,5, (IY+nn) FDCBnnE8	SET C,5, (IY+nn) FDCBnnE9	SET D,5, (IY+nn) FDCBnnEA	SET E,5, (IY+nn) FDCBnnEB	SET H,5, (IY+nn) FDCBnnEC	SET L,5, (IY+nn) FDCBnnED	SET 5,(IY+d) 4 23 FDCBnnEE	SET A,5, (IY+nn) FDCBnnEF
F	SET B,6, (IY+nn) FDCBnnF0	SET C,6, (IY+nn) FDCBnnF1	SET D,6, (IY+nn) FDCBnnF2	SET E,6, (IY+nn) FDCBnnF3	SET H,6, (IY+nn) FDCBnnF4	SET L,6, (IY+nn) FDCBnnF5	SET 6,(IY+d) 4 23 FDCBnnF6	SET A,6, (IY+nn) FDCBnnF7	SET B,7, (IY+nn) FDCBnnF8	SET C,7, (IY+nn) FDCBnnF9	SET D,7, (IY+nn) FDCBnnFA	SET E,7, (IY+nn) FDCBnnFB	SET H,7, (IY+nn) FDCBnnFC	SET L,7, (IY+nn) FDCBnnFD	SET 7,(IY+d) 4 23 FDCBnnFE	SET A,7, (IY+nn) FDCBnnFF