Extended Z80 instruction set

This is a general list of Z80 and Z80N instructions with descriptions. The instructions specific to Z80N have the encodings described first.

For a table of summaries without additional descriptions see the Z80 Instruction Table. There's also an external Z80N opcode value matrix on specnext.dev (https://table.specnext.dev/).

Z80N instructions opcodes

This is a table of the instructions and the encodings of all **Next-specific** Z80 instructions:

Instruction	Encoding (2-4 bytes) (hex)	T-States	Description	4 letter
`SWAPNIB`	`ED 23`	8	Swap the nibbles (4-bits) of A	SWAP
`MIRROR A`	`ED 24`	8	Mirror bits 0..7 to 7..0 of A	MIRR
`TEST $im8`	`ED 27` value	11	Change flags as AND A but A stays unaffected	TEST
`BSLA DE,B`	`ED 28`	8	Shift left DE for B bits	BSLA
`BSRA DE,B`	`ED 29`	8	Shift right signed DE for B bits	BSRA
`BSRL DE,B`	`ED 2A`	8	Shift right DE for B bits	BSRL
`BSRF DE,B`	`ED 2B`	8	Shift right DE, filling 1 for B bits	BSRF
`BRLC DE,B`	`ED 2C`	8	Rotate left DE for B bits	BRLC
`MUL D,E`	`ED 30`	8	Multiply D with E, result in DE	MUL
`ADD HL,A`	`ED 31`	8	Add unsigned A to HL	ADD
`ADD DE,A`	`ED 32`	8	Add unsigned A to DE	ADD
`ADD BC,A`	`ED 33`	8	Add unsigned A to BC	ADD
`ADD HL,$im16`	`ED 34 low high`	16	Add a constant to HL	ADD
`ADD DE,$im16`	`ED 35 low high`	16	Add a constant to DE	ADD
`ADD BC,$im16`	`ED 36 low high`	16	Add a constant to BC	ADD
`PUSH $im16`	`ED 8A high low`	23	Push a constant	PUSH
`OUTINB`	`ED 90`	16	OUTI, but don't change B	OTIB
`NEXTREG $im8,$im8`	`ED 91 register value`	20	Write v to Next register r	NREG
`NEXTREG $im8,A`	`ED 92 register`	17	Write A to Next register r	NREG
`PIXELDN`	`ED 93`	8	HL as a pixel address moved one line down	PXDN
`PIXELAD`	`ED 94`	8	HL := as ULA pixel address from D==Y E==X	PXAD
`SETAE`	`ED 95`	8	Get A as a mask for ULA pixel from E	STAE
`JP (C)`	`ED 98`	13	IN (C) then jump to a 64 byte section	JP
`LDIX`	`ED A4`	16	Extended LDI byte copy	LDIX
`LDWS`	`ED A5`	14	Byte copy as for Layer 2	LDWS
`LDDX`	`ED AC`	16	Extended LDD byte copy	LDDX
`LDIRX`	`ED B4`	21/16	Extended LDIR	LIRX
`LDPIRX`	`ED B7`	21/16	Byte copy as for a pattern fill	LPRX
`LDDRX`	`ED BC`	21/16	Extended LDDR	LDRX

More details and the explanations of all Z80 instructions implemented in the Next Z80 CPU follow.

Notes:

- The encoding of the operand of the `PUSH $im16` is unique: it is the only operand encoded as big-endian.
- The timings (T-States) are based on some limited testing and may not be accurate.

- The shorter aliases of the official mnemonics (the 4-letter column) are those suggested in an initiative led by Matt Davies, the author of Odin assembler, editor and debugger tool for ZX Next.
- A relevant early source for this table could be a Christmas 2018 edit of a comment in a discussion of an issue 312 of z88dk (https://github.com/z88dk/z88dk/issues/312#issuecomment-322659205).

Notation

- In the descriptions *any 8-bit register* means A, B, C, D, E, H, and L. The registers named F, I and R aren't a part of this set even if they are technically 8 bit registers. Additionally, IXH, IXL, IYH, IYL, which are the high and low byte parts of IX and IY, can be used in some of the instructions as 8-bit registers although this wasn't documented for the original Z80.
- IXY means IX or IY.
- For the status field:
  - S means Standard. It's in the Z80 manual. Everything should support it.
  - U means Undocumented. It works on Z80 chips, but it's not in the manual. These have been known for years and were acknowledged by Zilog, so they should work on everything, but some assemblers may vary the syntax.
  - E means Extension. It *only* works on the Z80 core on the Next. It'll probably only be accepted by assemblers that have been updated specifically for the Next.
- Each of the flag effects is documented as follows:
  - - means the flag is unchanged.
  - 1 or 0 mean the flag is set or reset as appropriate.
  - ? means we don't know what effect the instruction has on the flag.
  - ! means the instruction has an unusual effect on the flag which is documented in the description.
  - S means the effect on the flag is "standard". C is set if a carry/borrow occurred beyond the MSB; Z is set if the result of the operation is zero; H is set if a carry/borrow occurred beyond bit 3.
  - P, V, and L are used for the P/V flag which has several standard effects. P means it's parity. V means it's overflow. L means it checks BC as loop counter for some of the block copy and search instructions: P/V = (BC != 0)

Register and Data manipulation

LD (LoaD)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`LD r, r'`	Register	Register	S	-	-	-	-	-	-	4	r := r'
`LD r,n`	Register	Immediate	S	-	-	-	-	-	-	7	r := n
`LD r, (HL)`	Register	Indirect	S	-	-	-	-	-	-	7	r := HL*
`LD r, (IXY+d)`	Register	Indexed	S	-	-	-	-	-	-	19	r := (IXY+d)*
`LD (HL),r`	Indirect	Register	S	-	-	-	-	-	-	7	HL* := r
`LD (IXY+d),r`	Indexed	Register	S	-	-	-	-	-	-	19	(IXY+D)* := r
`LD (HL), n`	Indirect	Immediate	S	-	-	-	-	-	-	10	HL* := n
`LD (IXY+d), n`	Indexed	Immediate	S	-	-	-	-	-	-	19	(IXY+d)* := n
`LD A, (BC/DE)`	Accumulator	Indirect	S	-	-	-	-	-	-	7	A := rr*
`LD A, (nn)`	Accumulator	Address	S	-	-	-	-	-	-	13	A := (nn)*
`LD (BC/DE), A`	Indirect	Accumulator	S	-	-	-	-	-	-	7	rr* := A
`LD (nn), A`	Address	Accumulator	S	-	-	-	-	-	-	13	(nn)* := A
`LD A, I`	Accumulator	Register	S	-	0	!	0	S	S	9	A := I; P/V:=IFF2
`LD A, R`	Accumulator	Register	S	-	0	!	0	S	S	9	A := R; P/V:=IFF2
`LD I, A`	Register	Accumulator	S	-	-	-	-	-	-	9	I := A
`LD R, A`	Register	Accumulator	S	-	-	-	-	-	-	9	R := A
`LD BC/DE/HL/SP, nn`	Register	Immediate	S	-	-	-	-	-	-	10	rr := nn
`LD IXY, nn`	Register	Immediate	S	-	-	-	-	-	-	14	rr := nn
`LD HL, (nn)`	Register	Address	S	-	-	-	-	-	-	16	HL := (nn)*
`LD BC/DE/SP/IXY, (nn)`	Register	Address	S	-	-	-	-	-	-	20	rr := (nn)*
`LD (nn), HL`	Address	Register	S	-	-	-	-	-	-	16	(nn)* := HL
`LD (nn), BC/DE/SP/IXY`	Address	Register	S	-	-	-	-	-	-	20	(nn)* := rr
`LD SP, HL`	Register	Register	S	-	-	-	-	-	-	6	SP := HL
`LD SP, IXY`	Register	Register	S	-	-	-	-	-	-	10	SP := IXY

The basic data load/transfer instruction. Transfers data from the location specified by the second argument, to the location specified by the first. Available combinations are as follows:

- Any 8-bit register can be:
  - loaded with an immediate value;
  - loaded with the contents of any other 8-bit register except I and R;
  - loaded with the contents of, or stored in, memory pointed to by HL;
  - loaded with the contents of, or stored in, memory offset-indexed by IX or IY.
- Additionally, the accumulator A (only) can be:
  - loaded with the contents of, or stored in, memory pointed to by BC or DE;
  - loaded with the contents of, or stored in, memory pointed to by an immediate address;
  - loaded with the contents of I or R.
- Any 16-bit register pair can be:
  - loaded with an immediate value;
  - loaded with the contents of, or stored in, memory pointed to by an immediate address.
- Additionally, SP (only) can be:
  - loaded with the contents of HL, IX, or IY.
  - The planned **ld hl, sp** didn't make it to Next yet, one possible workaround is: **ld hl,0**; **add hl,sp**;
- Memory referred to by HL or through IX can be assigned immediate values.

Although 16-bit register pairs cannot be directly moved between each other, they can be moved by moving the two 8-bit registers. (SP gets a special case because it can't be addressed via 8-bit registers.) Some assemblers will provide built-in macro instructions allowing, for example, **ld bc, de**.

LD instructions do not alter any flags unless I or R are loaded into A.

EX (EXchange)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`EX DE, HL`	Register	Register	S	-	-	-	-	-	-	4	swap(DE,HL)
`EX AF, AF'`	Register	Register	S	!	!	!	!	!	!	4	swap(AF,AF')
`EX (SP), HL`	Indirect	Register	S	-	-	-	-	-	-	19	swap(SP*,HL)
`EX (SP), IXY`	Indirect	Register	S	-	-	-	-	-	-	23	swap(SP*,IXY)

Exchanges the contents of two sources. The only permitted combinations are

- Exchanging DE and HL;
- Exchanging AF and AF';
- Exchanging HL, IX, or IY with the contents of memory pointed to by SP.

EXX (EXchange all)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`EXX`	-	-	S	-	-	-	-	-	-	4	swap(BC,BC');swap(DE,DE');swap(HL,HL')

Exchanges BC, DE, and HL with their shadow registers. AF and AF' are not exchanged.

PUSH

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`PUSH BC/DE/HL/AF`	Register	-	S	-	-	-	-	-	-	11	SP-=2; SP*:=rr
`PUSH IXY`	Register	-	S	-	-	-	-	-	-	15	SP-=2; SP*:=rr
`PUSH nn`	Immediate	-	E	-	-	-	-	-	-	23	SP-=2; SP*:=nn

Pushes the given argument on the stack. This can be any 16-bit register pair except SP. SP is reduced by 2 and then the argument is written to the new location SP points to.

POP

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`POP BC/DE/HL`	Register	-	S	-	-	-	-	-	-	10	rr:=SP*; SP+=2
`POP AF`	Register	-	S	!	!	!	!	!	!	10	rr:=SP*; SP+=2
`POP IXY`	Register	-	S	-	-	-	-	-	-	14	rr:=SP*; SP+=2

Pops the given argument from the stack. This can be any 16-bit register pair except SP. The current value at SP is copied to the register pair and then SP is raised by 2.

Popping into AF does set value of flag register F directly to low 8 bits of value from stack.

Block Copy

LDI (LoaD and Increment)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`LDI`	-	-	S	-	0	L	0	-	-	16	DE*:=HL*; DE++; HL++; BC--

Copies the byte pointed to by HL to the address pointed to by DE, then adds 1 to DE and HL and subtracts 1 from BC. P/V is set to (BC!=0), i.e. set when non zero.

LDIR (LoaD and Increment Repeated)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`LDIR`	-	-	S	-	0	L	0	-	-	21x+16	do LDI while(BC>0)

Automatically loops LDI until BC reaches zero. Note the last iteration starts when BC=1 and ends with BC=0 (starting the LDIR with BC=0 will start 64kiB transfer, most likely overwriting vital parts of system memory and/or code itself).

Flag effects are the same as LDI except that P/V will always be reset, because BC by definition reaches 0 before this instruction ends (normally - unless something overwrites LDIR opcode while BC>0).

Interrupts may interrupt LDIR instruction while looping (after each single LDI sub-part finished) and LDIR will resume after and finish loop properly.

LDD (LoaD and Decrement)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`LDD`	-	-	S	-	0	L	0	-	-	16	DE*:=HL*; DE--; HL--; BC--

Same as LDI, but subtracts 1 from DE and HL instead of adding.

LDDR (LoaD and Decrement Repeated)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`LDDR`	-	-	S	-	0	0	0	-	-	21x+16	do LDD while(BC>0)

Same as LDIR but loops LDD instead of LDI.

LDWS

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`LDWS`	-	-	E	-	0	!	S	S	S	14	DE*:=HL*; INC L; INC D;

Next-only extended opcode. Copies the byte pointed to by HL to the address pointed to by DE and increments only L and D. This is used for vertically copying bytes to the Layer 2 display.

The flags are identical to what the **INC D** instruction would produce.

Note the source data are read only from single 256B (aligned) block of memory, because only L is incremented, not HL.

LDIX, LDIRX, LDDX, LDDRX

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`LDIX`	-	-	E	-	-	-	-	-	-	16	{if HL*!=A DE*:=HL*;} DE++; HL++; BC--
`LDIRX`	-	-	E	-	-	-	-	-	-	21/16	do LDIX while(BC>0)
`LDDX`	-	-	E	-	-	-	-	-	-	16	{if HL*!=A DE*:=HL*;} DE++; HL--; BC--
`LDDRX`	-	-	E	-	-	-	-	-	-	21/16	do LDDX while(BC>0)

Next-only extended opcodes. Behave similarly as their non-X equivalents except the byte is not copied if it is equal to A and LDDX/LDDRX advance DE by incrementing it (like LDI), while HL is decremented (like LDD).

Second difference to non-X instructions (as usual with next-only opcodes due to implementation), the extended ones don't modify any flags.

LDPIRX

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`LDPIRX`	-	-	E	-	-	-	-	-	-	21/16	do{t:=(HL&$FFF8+E&7)*; {if t!=A DE*:=t;} DE++; BC--} while(BC>0)

Similar to LDIRX except the source byte address is not just HL, but is obtained by using the top 13 bits of HL and the lower 3 bits of DE and HL does not increment during whole loop (HL works as base address of aligned 8 byte lookup table, DE works as destination and also wrapping index 0..7 into table). This is intended for "pattern fill" functionality.

Block Search

CPI (ComPare and Increment)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`CPI`	-	-	S	-	1	L	S	!	S	16	HL*==A?; HL++; BC--

Compares the byte pointed to by HL with the contents of A and sets Z if it matches or S if the comparison goes negative. Then adds 1 to HL and subtracts 1 from BC. P/V is set to (BC!=0), i.e. set when non zero.

CPIR (ComPare and Increment Repeated)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`CPIR`	-	-	S	-	1	L	S	!	S	21x+16	do CPI while (!Z && BC>0)

Automatically loops CPI until either Z is set (A is found in the byte pointed to by HL) or P/V is reset (BC reached 0).

Flag effects are the same as CPI except that one of the two terminating flag conditions will always be true.

CPD (ComPare and Decrement)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`CPD`	-	-	S	-	1	L	S	!	S	16	HL*==A?; HL--; BC--

Same as CPI, but subtracts 1 from HL instead of adding it.

CPDR (ComPare and Decrement Repeated)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`CPDR`	-	-	S	-	1	L	S	!	S	21x+16	do CPD while (!Z && BC>0)

Same as CPIR but loops CPD instead of CPI.

Arithmetic

ADD

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`ADD A, r`	Accumulator	Register	S	S	0	V	S	S	S	4	A+=r
`ADD A, n`	Accumulator	Immediate	S	S	0	V	S	S	S	7	A+=n
`ADD A, (HL)`	Accumulator	Indirect	S	S	0	V	S	S	S	7	A+=HL*
`ADD A, (IXY+d)`	Accumulator	Indexed	S	S	0	V	S	S	S	19	A+=(IXY+d)*
`ADD HL, BC/DE/HL/SP`	Register	Register	S	S	0	-	!	-	-	11	HL+=rr
`ADD IXY, BC/DE/IXY/SP`	Register	Register	S	S	0	-	!	-	-	15	IXY+=rr
`ADD HL/DE/BC, A`	Register	Register	E	?	-	-	-	-	-	8	rr+=unsigned A
`ADD HL/DE/BC, nn`	Register	Immediate	E	-	-	-	-	-	-	16	rr+=nn

Adds values together. Legal combinations are:

- When adding 8-bit values the first parameter must be A and the second may be:
  - The contents of an 8-bit register;
  - An immediate value;
  - The contents of memory pointed to by HL or by indexing based on IX or IY.
- When adding 16-bit values the first parameter must be HL, IX or IY and the second must be another 16-bit register pair. If the first parameter is IX or IY, the second cannot be HL or the other index register.
- For 16 bit additions (regular Z80), H is set if a carry occurred in bit 11 (ie, a half carry in the high byte)
- On the Spectrum Next the extended opcodes also allow the first parameter to be HL, DE, or BC and the second to be A (A will be zero extended to 16 bits) or an 16bit immediate value.

ADC (ADd with Carry)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`ADC A, r`	Accumulator	Register	S	S	0	V	S	S	S	4	A+=r+(CF?1:0)
`ADC A, n`	Accumulator	Immediate	S	S	0	V	S	S	S	7	A+=n+(CF?1:0)
`ADC A, (HL)`	Accumulator	Indirect	S	S	0	V	S	S	S	7	A+=HL*+(CF?1:0)
`ADC A, (IXY+d)`	Accumulator	Indexed	S	S	0	V	S	S	S	19	A+=(IXY+d)*+(CF?1:0)
`ADC HL, BC/DE/HL/SP`	Register	Register	S	S	0	V	!	S	S	15	HL+=rr+(CF?1:0)

Adds values together, adding an additional 1 if Carry is set. Legal combinations are the same as for ADD, although there are no extended opcode versions of ADC and in 16-bit values the first parameter can only be HL. For 16-bit values the H flag is set if carry from bit 11; otherwise, it is reset.

SUB

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`SUB r`	Register	-	S	S	1	V	S	S	S	4	A -= r
`SUB n`	Immediate	-	S	S	1	V	S	S	S	7	A -= n
`SUB (HL)`	Indirect	-	S	S	1	V	S	S	S	7	A -= HL*
`SUB (IXY+d)`	Indexed	-	S	S	1	V	S	S	S	19	A -= (IXY+d)*

Subtracts a value from A. Legal combinations are the same as for ADD for 8-bit, except that A does not need to be specified as the first parameter because subtraction can only be done from A. SUB cannot be used for 16-bit numbers.

SBC (SuBtract with Carry, er, borrow)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`SBC A, r`	Accumulator	Register	S	S	1	V	S	S	S	4	A-=(r+(CF?1:0))
`SBC A, n`	Accumulator	Immediate	S	S	1	V	S	S	S	7	A-=(n+(CF?1:0))
`SBC A, (HL)`	Accumulator	Indirect	S	S	1	V	S	S	S	7	A-=(HL*+(CF?1:0))
`SBC A, (IXY+d)`	Accumulator	Indexed	S	S	1	V	S	S	S	19	A-=((IXY+d)+(CF?1:0))
`SBC HL, BC/DE/HL/SP`	Register	Register	S	S	1	V	!	S	S	15	HL-=(rr+(CF?1:0))

Subtracts values, subtracting an additional 1 if Carry is set. Legal combinations are the same as for ADD, although there are no extended opcode versions of SBC and in 16-bit values the first parameter can only be HL. For 16-bit values the H flag is set if borrow from bit 12; otherwise, it is reset.

AND, OR, XOR

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`AND r`	Register	-	S	0	0	P	1	S	S	4	A := A & r
`AND n`	Immediate	-	S	0	0	P	1	S	S	7	A := A & n
`AND (HL)`	Indirect	-	S	0	0	P	1	S	S	7	A := A & HL*
`AND (IXY+d)`	Indexed	-	S	0	0	P	1	S	S	19	A := A & (IXY+d)*
`OR r`	Register	-	S	0	0	P	0	S	S	4	A := A OR r
`OR n`	Immediate	-	S	0	0	P	0	S	S	7	A := A OR n
`OR (HL)`	Indirect	-	S	0	0	P	0	S	S	7	A := A OR HL*
`OR (IXY+d)`	Indexed	-	S	0	0	P	0	S	S	19	A := A OR (IXY+d)*
`XOR r`	Register	-	S	0	0	P	0	S	S	4	A := A ^ r
`XOR n`	Immediate	-	S	0	0	P	0	S	S	7	A := A ^ n
`XOR (HL)`	Indirect	-	S	0	0	P	0	S	S	7	A := A ^ HL*
`XOR (IXY+d)`	Indexed	-	S	0	0	P	0	S	S	19	A := A ^ (IXY+d)*

Performs the appropriate bitwise operator on A. Legal combinations are the same as SUB.

XOR A is faster and shorter than LD A,0

MIRROR

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`MIRROR A`	Register	-	E	-	-	-	-	-	-	8	A[76543210]:=A[01234567]

Next extended opcode. Mirrors (reverses the order) of bits in the accumulator. Older core versions supported MIRROR DE, but this was removed.

CP (ComPare)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`CP r`	Register	-	S	S	1	V	S	S	S	4	A-=r?
`CP n`	Immediate	-	S	S	1	V	S	S	S	7	A-=n?
`CP (HL)`	Indirect	-	S	S	1	V	S	S	S	7	A-=HL*?
`CP (IXY+d)`	Indexed	-	S	S	1	V	S	S	S	19	A-=(IXY+d)?

Sets the flags as if a SUB was performed but does not perform it. Legal combinations are the same as SUB. This is commonly used to set the flags to perform an equality or greater/less test.

- CP is *not* equivalent to "if" in high level languages. Flag based jumps can follow any instruction that sets the flags, not just CP.

TEST

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`TEST n`	Immediate	-	E	S	?	P	S	S	S	11	A&n?

Next extended opcode. Similar to CP, but performs an AND instead of a subtraction.

INC (INCrement)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`INC r`	Register	-	S	-	0	!	S	S	S	4	r++
`INC (HL)`	Indirect	-	S	-	0	!	S	S	S	11	HL*++
`INC (IXY+d)`	Indexed	-	S	-	0	!	S	S	S	23	(IXY+d)*++
`INC BC/DE/HL/SP`	Register	-	S	-	-	-	-	-	-	6	rr++
`INC IXY`	Register	-	S	-	-	-	-	-	-	10	rr++

Increments the target by one. The argument can be any 8-bit register, any 16-bit register pair, or the address pointed to by HL or indexed via IX or IY. P/V is set if the target held $7F. N is reset.

- INC A is faster than ADD 1.

DEC (DECrement)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`DEC r`	Register	-	S	-	1	!	S	S	S	4	r--
`DEC (HL)`	Indirect	-	S	-	1	!	S	S	S	11	HL*--
`DEC (IXY+d)`	Indexed	-	S	-	1	!	S	S	S	23	(IXY+D)*--
`DEC BC/DE/HL/SP`	Register	-	S	-	-	-	-	-	-	6	rr--
`DEC IXY`	Register	-	S	-	-	-	-	-	-	10	rr--

Decrements the target by one. Allowed arguments are the same as INC. Flag effects are the same as INC except H refers to borrow, not carry; and P/V is set if the target held $80.

- DEC A is faster than SUB 1.

RLC (Rotate Left Circular)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`RLC r`	Register	-	S	!	0	P	0	S	S	8	x:=r[7]; r:=r<<1; r[0]:=x; CF:=x
`RLC (HL)`	Indirect	-	S	!	0	P	0	S	S	15	x:=HL*[7]; HL*:=HL*<<1; HL*[0]:=x; CF:=x
`RLC (IXY+d)`	Indexed	-	S	!	0	P	0	S	S	23	x:=(IXY+d)*[7]; (IXY+d)*:=(IXY+d)*<<1; (IXY+d)*[0]:=x; CF:=x
`RLC r, (IX+d)`	Register	Indexed	U	!	0	P	0	S	S	23	x:=(IX+d)*[7]; (IX+d)*:=(IX+d)*<<1; (IX+d)*[0]:=x; CF:=x; r:=(IX+d)*

Rotates the target bitwise left by one position. The MSB is copied to bit 0, and also to Carry. Can be applied to any 8-bit register or to a location in memory pointed to by HL or indexed via IX or IY. The final alternative is undocumented, and stores the result in a register as well as performing the operation.

RL (Rotate Left)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`RL r`	Register	-	S	!	0	P	0	S	S	8	x:=r[7]; r:=r<<1; r[0]:=CF; CF:=x
`RL (HL)`	Indirect	-	S	!	0	P	0	S	S	15	x:=HL*[7]; HL*:=HL*<<1; HL*[0]:=CF; CF:=x
`RL (IXY+d)`	Indexed	-	S	!	0	P	0	S	S	23	x:=(IXY+d)*[7]; (IXY+d)*:=(IXY+d)*<<1; (IXY+d)*[0]:=CF; CF:=x
`RL r, (IX+d)`	Register	Indexed	U	!	0	P	0	S	S	23	x:=(IX+d)*[7]; (IX+d)*:=(IX+d)*<<1; (IX+d)*[0]:=CF; CF:=x; r:=(IX+d)*

Same as RLC, except the MSB is copied to Carry only, and the previous contents of Carry are copied to bit 0.

RRC, RR (Rotate Right Circular, Rotate Right)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`RRC r`	Register	-	S	!	0	P	0	S	S	8	x:=r[0]; r:=r>>1; r[7]:=x; CF:=x
`RRC (HL)`	Indirect	-	S	!	0	P	0	S	S	15	x:=HL*[0]; HL*:=HL*>>1; HL*[7]:=x; CF:=x
`RRC (IXY+d)`	Indexed	-	S	!	0	P	0	S	S	23	x:=(IXY+d)*[0]; (IXY+d)*:=(IXY+d)*>>1; (IXY+d)*[7]:=x; CF:=x
`RRC r,(IX+d)`	Register	Indexed	U	!	0	P	0	S	S	23	x:=(IX+d)*[0]; (IX+d)*:=(IX+d)*>>1; (IX+d)*[7]:=x; CF:=x; r:=(IX+d)*
`RR r`	Register	-	S	!	0	P	0	S	S	8	x:=r[0]; r:=r>>1; r[7]:=CF; CF:=x
`RR (HL)`	Indirect	-	S	!	0	P	0	S	S	15	x:=HL*[0]; HL*:=HL*>>1; HL*[7]:=CF; CF:=x
`RR (IXY+d)`	Indexed	-	S	!	0	P	0	S	S	23	x:=(IXY+d)*[0]; (IXY+d)*:=(IXY+d)*>>1; (IXY+d)*[7]:=CF; CF:=x
`RR r,(IX+d)`	Register	Indexed	U	!	0	P	0	S	S	23	x:=(IX+d)*[0]; (IX+d)*:=(IX+d)*>>1; (IX+d)*[7]:=CF; CF:=x; r=(IX+d)*

Same as RLC and RL except they rotate right instead of left.

SLA (Shift Left Arithmetic)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`SLA r`	Register	-	S	!	0	P	0	S	S	8	r:=r<<1
`SLA (HL)`	Indirect	-	S	!	0	P	0	S	S	15	HL*:=HL*<<1
`SLA (IXY+d)`	Indexed	-	S	!	0	P	0	S	S	23	(IXY+d)*:=(IXY+d)*<<1
`SLA r,(IX+d)`	Register	Indexed	U	!	0	P	0	S	S	23	(IX+d)*:=(IX+d)*<<1; r=(IX+d)*

Same as RL except bit 0 is set to zero, not the previous contents of Carry.

SRA (Shift Right Arithmetic)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`SRA r`	Register	-	S	!	0	P	0	S	S	8	r:=r>>1 OR r[7]
`SRA (HL)`	Indirect	-	S	!	0	P	0	S	S	15	HL*:=HL*>>1 OR HL*[7]
`SRA (IXY+d)`	Indexed	-	S	!	0	P	0	S	S	23	(IXY+d)*:=(IXY+d)*>>1 OR (IXY+d)*[7]
`SRA r,(IX+d)`	Register	Indexed	U	!	0	P	0	S	S	23	(IX+d)*:=(IX+d)*>>1 OR (IX+d)*[7]; r:=(IX+d)*

Same as RR except the MSB is left unchanged (on the assumption that it's the sign bit), not replaced with the previous contents of Carry.

SRL (Shift Right Logical)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`SRL r`	Register	-	S	!	0	P	0	S	0	8	r:=unsigned(r)>>1
`SRL (HL)`	Indirect	-	S	!	0	P	0	S	0	15	HL*:=unsigned(HL*)>>1
`SRL (IXY+d)`	Indexed	-	S	!	0	P	0	S	0	23	(IXY+d)*:=unsigned((IXY+d)*)>>1
`SRL r,(IXY+d)`	Register	Indexed	U	!	0	P	0	S	0	23	(IXY+d)*:=unsigned((IXY+d)*)>>1; r:=(IXY+d)*

Same as SLA except it shifts right instead of left.

RLCA, RLA, RRCA, RRA

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`RLCA`	-	-	S	!	0	-	0	-	-	4	x:=A[7]; A:=A<<1; A[0]:=x; CF:=x
`RLA`	-	-	S	!	0	-	0	-	-	4	x:=A[7]; A:=A<<1; A[0]:=CF; CF:=x
`RRCA`	-	-	S	!	0	-	0	-	-	4	x:=A[0]; A:=A>>1; A[7]:=x; CF:=x
`RRA`	-	-	S	!	0	-	0	-	-	4	x:=A[0]; A:=A>>1; A[7]:=CF; CF:=x

Same as their matching instruction except they work only on A, are faster, and do not alter S, Z or P/V.

SLL (Shift Left Logical)

This mnemonic has no associated opcode. There is no difference between a logical and arithmetic shift left, so both can use SLA, but some assemblers will allow SLL as an equivalent. Unfortunately, some will also assemble it as SL1. So it's probably worth just avoiding.

SL1 or SLI (Shift Left and Add 1) or (Shift Left and Increment)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`SL1 r`	Register	-	U	!	0	P	0	S	S	8	r:=(r<<1)+1
`SL1 (HL)`	Indirect	-	U	!	0	P	0	S	S	15	HL*:=(HL*<<1)+1
`SL1 (IXY+d)`	Indexed	-	U	!	0	P	0	S	S	23	(IXY+d)*:=((IXY+d)*<<1)+1
`SL1 r,(IXY+d)`	Register	Indexed	U	!	0	P	0	S	S	23	(IXY+d)*:=((IXY+d)*<<1)+1; r=(IXY+d)*

Undocumented opcodes that behave like SLA, but set bit 0 to 1 instead of 0.

RLD (Rotate Left bcd Digit)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`RLD`	-	-	S	-	0	P	0	S	S	18	x=HL*; HL*[0123]:=A[0123]; HL*[7654]:=x[0123]; A[0123]:=x[7654]

Rotates the lower nibble of the byte pointed to by HL, the upper nibble of that byte, and the lower nibble of A, in that order.

RRD (Rotate Right bcd Digit)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`RRD`	-	-	S	-	0	P	0	S	S	18	x=HL*; HL*[7654]:=A[0123]; HL*[0123]:=x[7654]; A[0123]:=x[0123]

Same as RLD, but the order is: the lower nibble pointed to by HL, the lower nibble of the A, and the upper nibble of the byte.

Barrel (variable amount) shift and rotate (cores v2+ only)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`BSLA DE,B`	-	-	E	-	-	-	-	-	-	DE:=DE<<(B&31)	8
`BSRA DE,B`	-	-	E	-	-	-	-	-	-	DE:=signed(DE)>>(B&31)	8
`BSRL DE,B`	-	-	E	-	-	-	-	-	-	DE:=unsigned(DE)>>(B&31)	8
`BSRF DE,B`	-	-	E	-	-	-	-	-	-	DE:=~(unsigned(~DE)>>(B&31))	8
`BRLC DE,B`	-	-	E	-	-	-	-	-	-	DE:=DE<<(B&15) OR DE>>(16-B&15)	8

Shift instructions use only bits 4..0 of B, BSLA shifts DE left, BSRA/BSRL/BSRF shifts DE right in arithmetic/logical/fill-one way. BRLC rotates DE left by B places, uses only bits 3..0 of B (to rotate right, use B=16-places).

If you are implementing BSRF in C (or similar), be careful with implicit variable type promotion. You will need to do something like this: DE:=~((uint16_t)~DE>>(B&31))

CPL (ComPLement)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`CPL`	-	-	S	-	1	-	1	-	-	4	A:=~A

Inverts the contents of the accumulator.

NEG (NEGate)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`NEG`	-	-	S	!	1	!	S	S	S	8	A:=0-A

Subtracts the contents of the accumulator from zero, making it negative for the purpose of two's complement. P/V is set if A previously held $80. C is set if accumulator did not previously hold $00.

CCF (Complement Carry Flag)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`CCF`	-	-	S	!	0	-	!	-	-	4	CF:=!CF

Inverts the carry flag. (Does not, as might be assumed, clear it!) Also sets H to the previous value of the carry flag.

SCF (Set Carry Flag)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`SCF`	-	-	S	1	0	-	0	-	-	4	CF:=1

Sets the carry flag.

BIT (test BIT)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`BIT b,r`	Immediate	Register	S	-	0	?	1	!	?	8	r[b]==1?
`BIT b,(HL)`	Immediate	Indirect	S	-	0	?	1	!	?	12	HL*[b]==1?
`BIT b,(IXY+d)`	Immediate	Indexed	S	-	0	?	1	!	?	20	(IXY+d)*[b]==1?

Tests if a bit is set on target value. The first parameter states which bit. The second can be any 8-bit register, or the location in memory pointed to by HL or indexed by IX or IY. Sets Z if specified bit was 0. S and P/V are destroyed.

SET (SET bit)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`SET b,r`	Immediate	Register	S	-	-	-	-	-	-	8	r:=r OR (1<<b)
`SET b,(HL)`	Immediate	Register	S	-	-	-	-	-	-	15	HL*:=HL* OR (1<<b)
`SET b,(IXY+d)`	Immediate	Indexed	S	-	-	-	-	-	-	23	(IXY+d)*:=(IXY+d)* OR (1<<b)
`SET r,b,(IX+d)`	Immediate	Indexed	U	-	-	-	-	-	-	23	(IX+d)*:=(IX+d)* OR (1<<b); r:=(IX+d)*
`SETAE`	-	-	E	-	-	-	-	-	-	8	A:=unsigned($80)>>(E&7)

Sets the numbered bit on target value. The possible targets are the same as BIT. The three parameter variant is undocumented and stores the result in a register as well as performing the SET.

SETAE is a Next extended opcode which takes the bit number to set from E (only the low 3 bits) and sets whole A to value of that bit, but counted from top to bottom (E=0 will produce A:=$80, E=7 will produce A:=$01). This works as pixel mask for ULA bitmap modes, when E is 0..255 x-coordinate.

RES (RESet bit)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`RES b,r`	Immediate	Register	S	-	-	-	-	-	-	8	r:=r & (~(1<<b))
`RES b,(HL)`	Immediate	Register	S	-	-	-	-	-	-	15	HL*:=HL* & (~(1<<b))
`RES b,(IXY+d)`	Immediate	Indexed	S	-	-	-	-	-	-	23	(IXY+d)*:=(IXY+d)* & (~(1<<b))
`RES r,b,(IX+d)`	Immediate	Indexed	U	-	-	-	-	-	-	23	(IX+d)*:=(IX+d)* & (~(1<<b)); r:=(IX+d)*

Resets the numbered bit on target value. The possible targets are the same as BIT.

DAA (Decimal Adjust Accumulator)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`DAA`	-	-	S	!	-	P	!	S	S	8	if(A&$0F>$09 or HF) A±=$06; if(A&$F0>$90 or CF) A±=$60 (± depends on NF)

Modifies the contents of the accumulator based on the flags left by previous operation to correct for binary coded decimal (BCD).

MUL (MULtiply)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`MUL d,e`	-	-	E	-	-	-	-	-	-	8	DE:=D*E

Next extended opcode. Multiplies D by E, storing 16 bit result into DE. Does not alter any flags (the opcode is not compatible with any of the R800/Z180/eZ80/... variants of MUL, it is solely Next specific).

SWAPNIB (SWAP NIBbles)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`SWAPNIB`	-	-	E	-	-	-	-	-	-	8	A:=A[3210]<<4 OR A[7654]>>4

Next extended opcode. Swaps the high and low nibbles of the accumulator.

PIXELAD (PIXEL ADdress)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`PIXELAD`	-	-	E	-	-	-	-	-	-	8	HL:=$4000+((D&$C0)<<5)+((D&$07)<<8)+((D&$38)<<2)+(E>>3)

Next extended opcode. Takes E and D as the X,Y coordinate of a point and calculates the address of the byte containing this pixel in the pixel area of standard ULA screen 0, storing it in HL.

PIXELDN (PIXEL DowN)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx

`PIXELDN`	-	-	E	-	-	-	-	-	-	8	if(HL&$0700!=$0700) HL+=256;
											else if(HL&$e0!=$e0) HL:=HL&$F8FF+$20;
											else HL:=HL&$F81F+$0800

Updates the address in HL to move down by one line of pixels.

Control Flow

JP (JumP)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`JP nn`	Address	-	S	-	-	-	-	-	-	10	PC:=nn
`JP (HL)`	Register	-	S	-	-	-	-	-	-	4	PC:=HL (not PC:=HL*)
`JP (IXY)`	Register	-	S	-	-	-	-	-	-	8	PC:=IXY (not PC:=IXY*)
`JP (C)`	Register	-	E	?	?	?	?	?	?	13	PC:=PC&$C000+IN(C)<<6

Jumps (sets the PC) to the given address. The address can be given immediately or read from HL, IX, or IY. Note that although the variants that use register pairs *look* like they are using indirect addressing, JP (HL) jumps to the address stored in the register HL, not the address stored at the address HL points to. The JP (C) sets bottom 14 bits of current PC* to value read from I/O port: PC[13:0] = (IN (C) << 6) (can be used to execute code block read from a disk stream) * "current PC" is address of next instruction after JP (C), as the PC is advanced by fetching op code from memory and is already advanced when execution happens - if the JP instruction resides at the very end of 16k memory block (..FE or ..FF address), then newly composed PC value will land into following 16k block.

JP cc (JumP conditionally)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`JP Z/NZ/NC/C/PO/PE/P/M, nn`	Address	-	S	-	-	-	-	-	-	10	if cc PC:=nn

Conditionally jumps (sets the PC) to the given address. The condition is set in terms of the flags: Zero (Z, NZ), Carry (C, NC), Parity (PO, PE), and Sign (P/M). The address can only be given immediately for a conditional jump.

JR (Jump Relative)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`JR nn`	Immediate	-	S	-	-	-	-	-	-	12	PC+=nn
`JR C/NC/Z/NZ, nn`	Immediate	-	S	-	-	-	-	-	-	12 ; 7 if not cc	if cc PC+=nn

Jumps to an alternate address by *adding* the parameter to the PC - in other words the parameter is an adjustment, not an absolute address. When used in an assembler with labels the syntax should be the same as JP. The JR address can only be given immediately. Conditions are legal, but only those based on carry and zero.

DJNZ (Decrement reg. b and Jump if Not Zero)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`DJNZ n`	Immediate	-	S	-	-	-	-	-	-	13	B--; if B!=0 PC+=nn

Decrements B then performs JR NZ.. to the given address. Used for encapsulating loops.

CALL

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`CALL nn`	Address	-	S	-	-	-	-	-	-	17	SP-=2; SP*:=PC; PC:=nn
`CALL Z/NZ/C/NC/PO/PE/P/M, nn`	Address	-	S	-	-	-	-	-	-	17 ; 10 if not cc	if cc {SP-=2; SP*:=PC; PC:=nn}

Like JP (including the ability to have conditions), but PUSHes the PC before jumping. Used for subroutine calls.

- If a subroutine ends with a CALL to another subroutine immediately before the RET, it is more efficient to JP to the other subroutine and allow its RET to return from the whole combination. This might make high-level programmers queasy but your compiler already does it (tail call optimization)

RET (RETurn)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`RET`	-	-	S	-	-	-	-	-	-	10	PC:=SP*; SP+=2
`RET Z/NZ/C/NC/PO/PE/P/M`	-	-	S	-	-	-	-	-	-	11 ; 5 if not cc	if cc {PC:=SP*; SP+=2}

POPs the PC from the stack, returning from a previous CALL. This can also accept the same conditions as JP.

RETI (RETurn from Interrupt)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`RETI`	-	-	S	-	-	-	-	-	-	14	PC:=SP*; SP+=2

Returns from an interrupt service routine (same as RET instruction, but also does signal to I/O device that the interrupt routine is completed).

RETN (RETurn from Non-maskable interrupt)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`RETN`	-	-	S	-	-	-	-	-	-	14	IFF1:=IFF2; PC:=SP*; SP+=2

Returns from a non-maskable interrupt service routine.

RST (ReSTart)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`RST n`	Immediate	-	S	-	-	-	-	-	-	11	CALL n

Performs a CALL to a routine located at one of eight fixed locations ($00, $08, ..., $38) in the zero page. This is located in ROM, and on the Spectrum only a limited number of values are useful to call built-in ROM routines.

NOP (No OPeration)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`NOP`	-	-	S	-	-	-	-	-	-	4	PC+=1

Does "nothing" (just usual housekeeping like refreshing memory and advancing program counter to next instruction).

HALT

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`HALT`	-	-	S	-	-	-	-	-	-	4	waits for interrupt

Suspends the CPU until an interrupt is received (maskable interrupts must be enabled to break the wait). While CPU is waiting for interrupt, the fake NOP instruction is being executed, to keep memory refresh going on.

DI (Disable Interrupts)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`DI`	-	-	S	-	-	-	-	-	-	4	IFF1:=0; IFF2:=0

Disables maskable interrupts.

EI (Enable Interrupts)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`EI`	-	-	S	-	-	-	-	-	-	4	IFF1:=1; IFF2:=1

Enables maskable interrupts (after next instruction, i.e. for example "EI RET" - the interrupt may happen only after RET instruction is finished (or "EI DI" pair of instructions will not allow any maskable interrupt to happen).

IM (Interrupt Mode)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`IM n`	Immediate	-	S	-	-	-	-	-	-	8	Interrupt mode:=n

Sets interrupt handling mode. The default for Next is 1. 2 is used for user defined interrupt service routines. IM 0 is useless on Next (and pretty much everything else, to be honest)

Input and Output

IN r, (c); OUT (c), r

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`IN r, (c)`	Register	Register	S	-	0	P	0	S	S	12	r := in(BC)
`OUT (c),r`	Register	Register	S	-	-	-	-	-	-	12	out(BC,r)

Inputs or outputs a byte value from the 16-bit port number given in BC. R can be any 8-bit register. Note that the port number is given in BC even though the instruction refers only to C. Some assemblers will allow the instruction to be written with "(bc)" instead of "(c)" as a reminder.

IN (c); OUT (c), 0

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`IN (c)`	Register	-	U	-	0	P	0	S	S	12	in(BC)?
`OUT (c),0`	Register	Immediate	U	-	-	-	-	-	-	12	out(BC,0)

Undocumented opcodes. The IN variation performs an input, but does not store the result, only setting the flags. The OUT variation outputs 0 on the port. This is the only number that can be output to a port in immediate mode. The Next FPGA does output zero, but some Z80 chips manufactured differently from early batches output different value like 255, so it is not recommended to use OUT (C),0 if you want to reuse your code also on classic ZX Spectrum or publish it as example.

IN a, (n); OUT (n), a

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`IN A, (n)`	Accumulator	Immediate	S	-	-	-	-	-	-	11	A := in(An)
`OUT (n),A`	Immediate	Accumulator	S	-	-	-	-	-	-	11	out(An,A)

Inputs or outputs the byte value in A from a 16-bit port number where the lower byte is N and the upper byte is A. This is only likely to be useful in cases where the upper byte of the port number is not relevant.

INI (INput and Increment)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`INI`	-	-	S	?	1	?	?	!	?	16	HL*:=in(BC); HL++; B--

Inputs a value from port BC into memory at the address stored in HL, then increments HL and decrements B. Because B is decremented and is part of the port number, the port number for future INI instructions will change unless it is reset. Z is set if B reached 0; C, S, H, and P/V are destroyed.

INIR (INput and Increment Repeated)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`INIR`	-	-	S	?	1	?	?	1	?	21x+16	do INI while(B>0)

Loops INIR until B reaches 0, so that if INIR starts with B=0 it loops 256 times. Because B is decremented during this process and is part of the port number, this is unlikely to be useful except when the upper byte of the port number is not relevant. Interrupts are recognized during execution.

IND, INDR (INput and Decrement, INput and Decrement Repeated)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`IND`	-	-	S	?	1	?	?	!	?	16	HL*:=in(BC); HL--; B--
`INDR`	-	-	S	?	1	?	?	1	?	21x+16	do IND while(B>0)

Behave like INI and INIR except that HL is decremented instead of incremented.

OUTI (Out and Increment), OTIR (Out and Increment Repeated), OUTD (Out and Decrement), OTDR (Out and Decrement Repeated)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`OUTI`	-	-	S	?	1	?	?	!	?	16	B--; out(BC,HL*); HL++
`OTIR`	-	-	S	?	1	?	?	1	?	21x+16	do OUTI while (B>0)
`OUTD`	-	-	S	?	1	?	?	!	?	16	B--; out(BC,HL*); HL--
`OTDR`	-	-	S	?	1	?	?	1	?	21x+16	do OUTD while (B>0)

Behave like INI, INIR, IND, INDR except that they output instead of input and B is decremented **before** the output instead of after. Condition check on B is performed **after**, so that if OTIR starts with B=0 it loops 256 times.

OUTINB (Out and Increment with No B)

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`OUTINB`	-	-	E	?	?	?	?	?	?	16	out(BC,HL*); HL++

Next extended opcode. Behaves like OUTI, but doesn't decrement B.

NEXTREG

Mnemonic	Addressing mode 1	Addressing mode 2	Status	C	N	PV	H	Z	S	Tstates	Shortfx
`NEXTREG n,n'`	Immediate	Immediate	E	-	-	-	-	-	-	20	HwNextReg_n:=n'
`NEXTREG n, A`	Immediate	Accumulator	E	-	-	-	-	-	-	17	HwNextReg_n:=A

Next extended opcode. Directly sets the Next Feature Control Registers without going through ports TBBlue_Register_Select and TBBlue Register Access.

A note on some Z80-N specific observations

2021-09-16: figuring out the hard way, the three Z80N instructions `ADD HL/DE/BC,A` actually do NOT preserve carry flag, but change it to undefined value (verified with core 3.1.5). There's also strong suspicion (but not verified yet), that LDIX/LDDX/LDIRX/LDDRX/LDPIRX do affects flags the same way as LDI/LDIR - to be verified.

2025-01-25: Testing 3.02.x 1) `LDIX`, `LDDX`, `LDIRX`, `LDDRX` affect the flags similarly to `LDI`, `LDD`, `LDIR` and `LDDR`. 2) `ADD HL,A`, `ADD DE,A` and `ADD BC,A` most probably always reset the carry flag.

Errata

This section lists the changes of the specification of the behavior of Z80N instructions, compared to the previous content of this wiki page.

2022-01-11: The previous versions of this page didn't match what was documented (http://www.z80.info/zip/z80-documented.pdf) for several years, that regular Z80 INI/IND/INIR/INDR/OUTI/OUTD/OTIR/OTDR instructions do modify carry flag (contrary to the official Z80 documentation and many Internet resources describing Z80 instructions).