THE Z80 INSTRUCTIONS: INDIVIDUAL DESCRIPTION

ABBREVIATIONS

FLAG	ON	OFF
Carry	C (carry)	NC (no carry)
Sign	M (minus)	P (plus)
Zero	Z (zero)	NZ (non zero)
Parity	PE (even)	PO (odd)

● changed functionally according to operation
O flag is set to zero
1 flag is set to one
? flag is set randomly by operation
X special case, see accompanying note on that page

bit positions 3 and 5 are always random

ADC A, s

Add accumulator and specified operand with carry.

Function: A ← A + s + C

Format: s: may be r, n, (HL),(IX + d), or (IY + d)

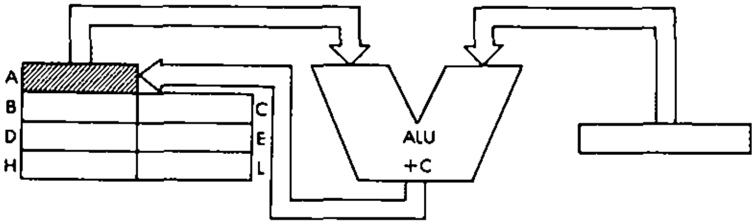
r	1	0	0	0	1	← r →			
n	1	1	0	0	1	1	1	0	byte 1: CE
	← n →								byte 2: immediate data
(HL)	1	0	0	0	1	1	1	0	8E
(IX + d)	1	1	0	1	1	1	0	1	byte 1: DD
	1	0	0	0	1	1	1	0	byte 2: 8E
	← d →								byte 3: offset value
(IY + d)	1	1	1	1	1	1	0	1	byte 1: FD
	1	0	0	0	1	1	1	0	byte 2: 8E
	← d →								byte 3: offset value

r may be any one of:

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

Description: The operand s and the carry flag C from the status register are added to the accumulator, and the result is stored in the accumulator. s is defined in the description of the similar ADD instructions.

Data Flow:



Timing:

s:	*M cycles:*	*T states:*	*usec @ 2 MHz:*
r	1	4	2
n	2	7	3.5
(HL)	2	7	3.5
(IX + d)	5	19	9.5
(IY + d)	5	19	9.5

Addressing Mode: r: implicit; n: immediate; (HL): indirect; (IX + d), (IY + d): indexed.

Byte Codes: ADC A,r

r:	A	B	C	D	E	H	L
	8F	88	89	8A	88	8C	8D

Flags:

S	Z		H		P/V	N	C
●	●		●		●	○	●

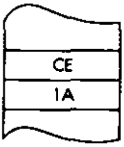
Example: ADC A, 1A

Before:

	A		F
A	06	13	F

After:

	A		F
A	21	10	F



OBJECT CODE

ADC HL, ss

Add with carry HL and register pair ss.

Function: HL ← HL + ss + C

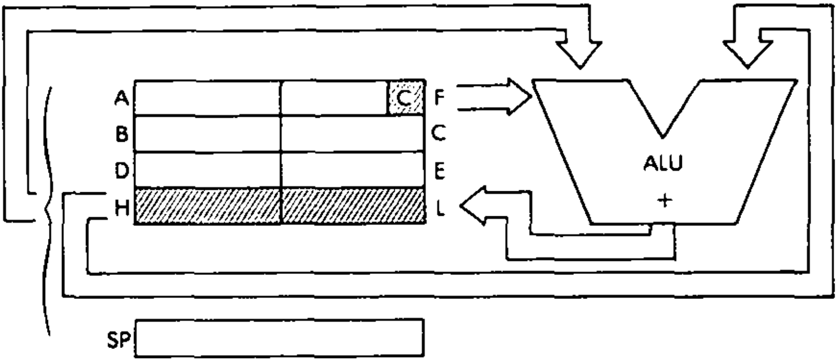
Format:

1	1	1	0	1	1	0	1	byte 1: ED
0	1	s	s	1	0	1	0	byte 2

Description: The contents of the HL register pair are added to the contents of the specified register pair, and then the contents of the carry flag are added. The final result is stored back in HL. ss may be any one of:

BC – 00	HL – 10
DE – 01	SP – 11

Data Flow:



Timing: 4 M cycles; 15 T states: 75 usec @ 2 MHz

Addressing Mode: Implicit.

Byte Codes: ss:

	BC	DE	HL	SP
ED-	4A	5A	6A	7A

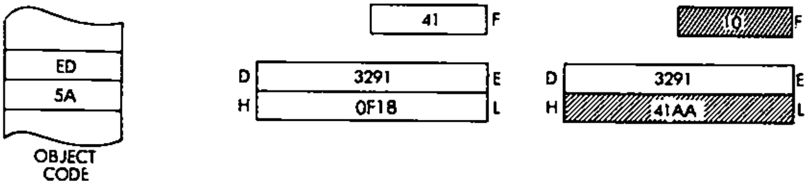
Flags:

S	Z		H		P/V	N	C
●	●		?		●	○	●

H is set if there is a carry from bit 11.

Example: ADC HL, DE

Before: After:



ADD A, (HL) Add accumulator with indirectly addressed memory location (HL).

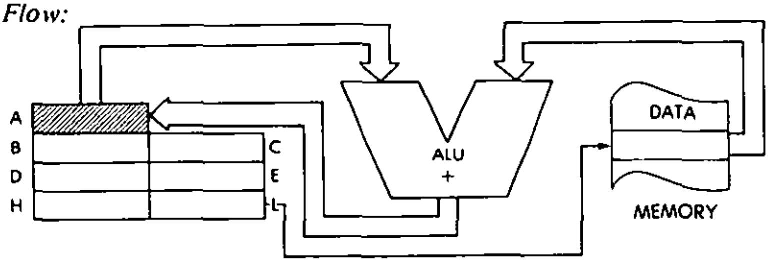
Function: A ← A + (HL)

Format:

1	0	0	0	0	1	1	0	86

Description: The contents of the accumulator are added to the contents of the memory location addressed by the HL register pair. The result is stored in the accumulator.

Data Flow:



Timing: 2 M cycles; 7 T states: 3.5 usec @ 2 MHz

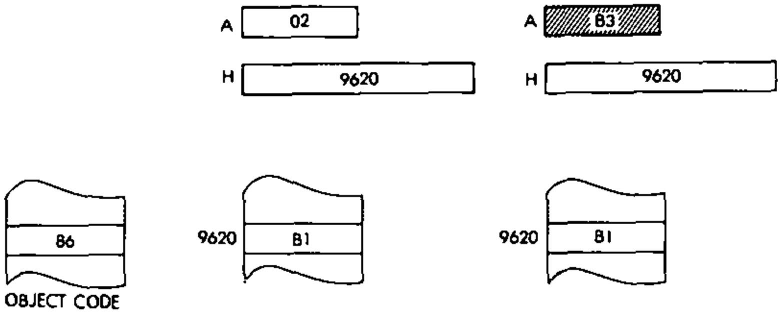
Addressing Mode: Indirect.

Flags:

S	Z		H		P/V	N	C
●	●		●		●	○	●

Example: ADD A, (HL)

Before: After:



ADD A, (IX + d) Add accumulator with indexed addressed memory location (IX + d)

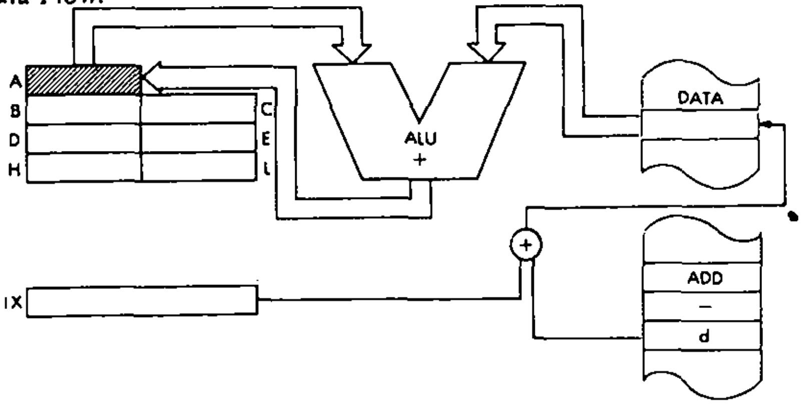
Function: A ← A + (IX + d)

Format:

1	1	0	1	1	1	0	1	byte 1: DD
1	0	0	0	0	1	1	0	byte 2: 86
←			d				→	byte 3: offset value

Description: The contents of the accumulator are added to the contents of the memory location addressed by the contents of the IX register plus the immediate offset value. The result is stored in the accumulator.

Data Flow:



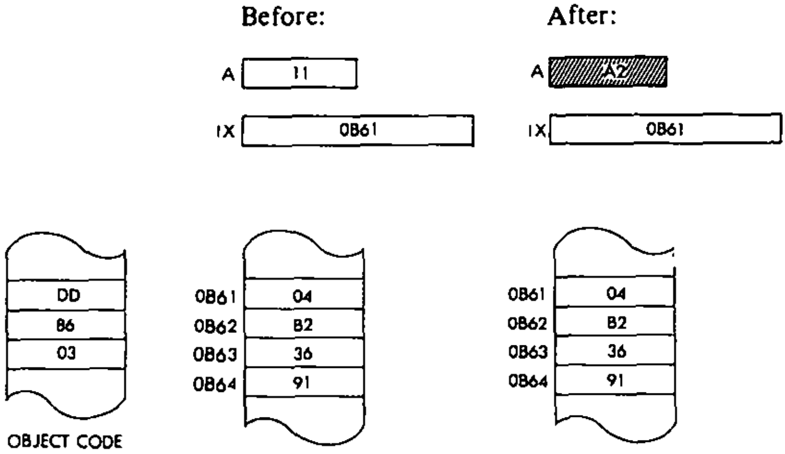
Timing: 5 M cycles; 19 T states: 9.5 usec @ 2 MHz

Addressing Mode: Indexed.

Flags:

S	Z		H		P/V	N	C
●	●		●		●	○	●

Example: ADD A, (IX + 3)



ADD A, (IY + d)

Add accumulator with indexed addressed memory location (IY + d)

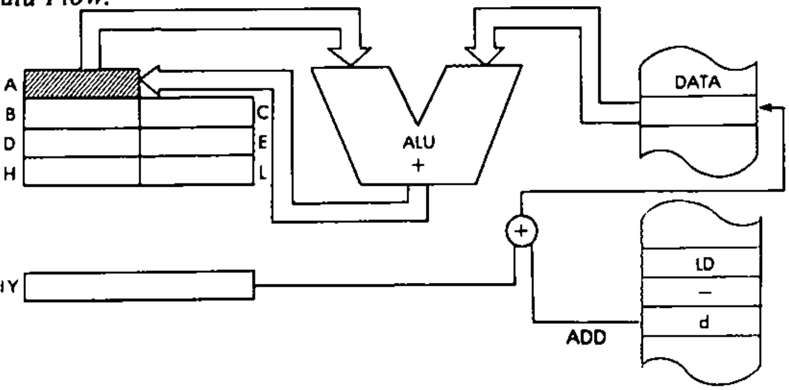
Function: A ← A + (IY + d)

Format:

1	1	1	1	1	1	0	1	byte 1: FD
1	0	0	0	0	1	1	0	byte 2: 86
← d →								byte 3: offset value

Description: The contents of the accumulator are added to the contents of the memory location addressed by the contents of the IY register plus the given offset value. The result is stored in the accumulator.

Data Flow:



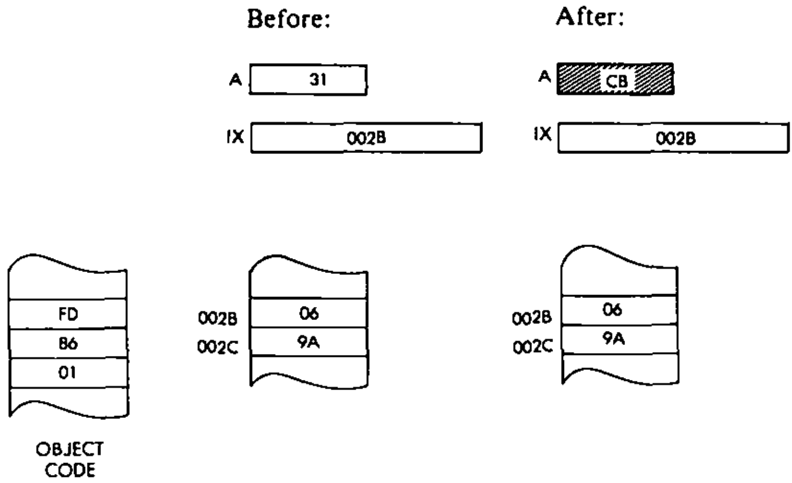
Timing: 5 M cycles; 19 T states; 9.5 usec @ 2 MHz

Addressing Mode: Indexed.

Flags:

S	Z		H		P/V	N	C
●	●		●		●	○	●

Example: ADD A, (IY + 1)



ADD A, n

Add accumulator with immediate data n.

Function: A ← A + n

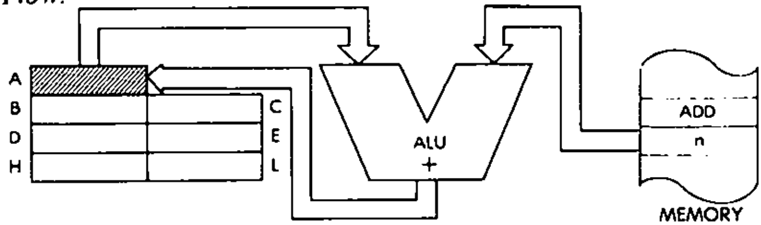
Format:

1	1	0	0	0	1	1	0	byte 1: C6

← n →	byte 2: immediate data

Description: The contents of the accumulator are added to the contents of the memory location immediately following the op code. The result is stored in the accumulator.

Data Flow:



Timing: 2 M cycles; 7 T states: 3.5 usec @ 2 MHz

Addressing Mode: Immediate.

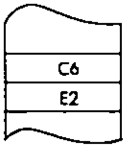
Flags:

S	Z		H		P/Ⓥ	N	C
●	●		●		●	○	●

Example: ADD A, E2

Before: A 43

After: A 25



OBJECT CODE

ADD A, r

Add accumulator with register r.

Function: A ← A + r

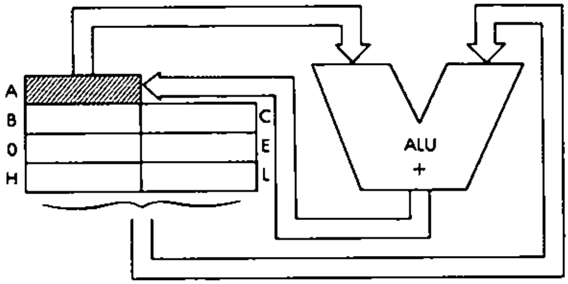
Format:

1	0	0	0	0	← r →

Description: The contents of the accumulator are added with the contents of the specified register. The result is placed in the accumulator. r may be any one of:

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

Data Flow:



Timing: 1 M cycle; 4 T states: 2 usec @ 2 MHz.

Addressing Mode: Implicit.

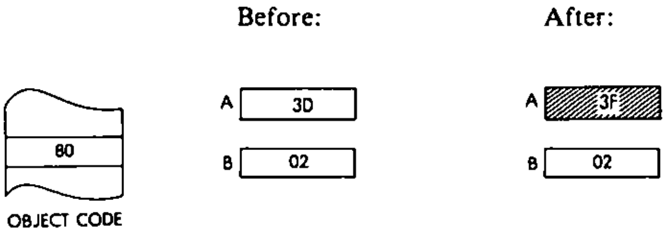
Byte Codes: r:

A	B	C	D	E	H	L
87	80	81	82	83	84	85

Flags:

S	Z		H		P/Ⓥ	N	C
●	●		●		●	○	●

Example: ADD A, B



ADD HL, ss

Add HL and register pair ss.

Function: HL ← HL + ss

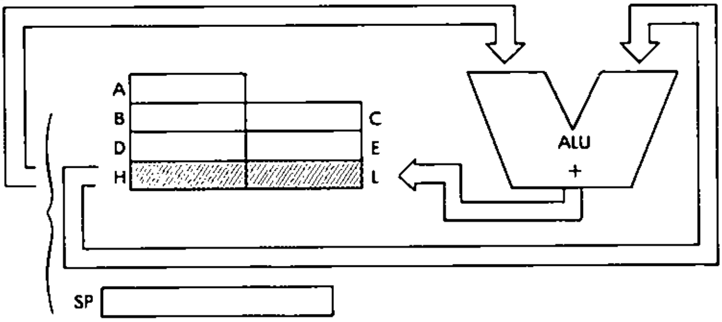
Format:

0	0	s	s	1	0	0	1

Description: The contents of the specified register pair are added to the contents of the HL register pair and the result is stored in HL. ss may be any one of:

BC – 00	HL – 10
DE – 01	SP – 11

Data Flow:



Timing: 3 M cycles; 11 T states: 5.5 usec @ 2 MHz

Addressing Mode: Implicit.

Byte Codes: ss:

BC	DE	HL	SP
09	19	29	39

Flags:

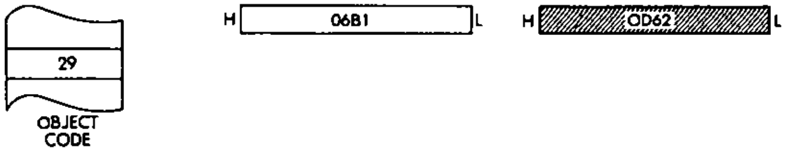
S	Z		H		P/V	N	C
			?			○	●

C is set by carry from bit 15, reset otherwise.

H is set by a carry from bit 11

Example: ADD HL, HL

Before: After:



ADD IX, rr

Add IX with register pair rr.

Function: IX ← IX + rr

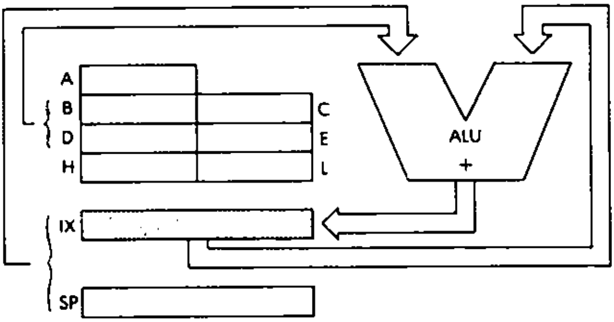
Format:

1	1	0	1	1	1	0	1	byte 1: DD
0	0	r	r	1	0	0	1	byte 2

Description: The contents of the IX register are added to the contents of the specified register pair and the result is stored back in IX. rr may be anyone of:

BC – 00	IX – 10
DE – 01	SP – 11

Data Flow:



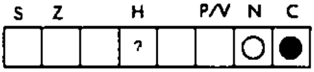
Timing: 4 M cycles; 15 T states: 7.5 usec @ 2 MHz

Addressing Mode: Implicit.

Byte Codes:

rr:	BC	DE	IX	SP
DD-	09	19	29	39

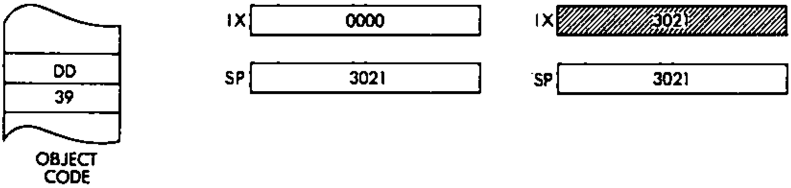
Flags:



H is set by carry out of bit 11.
C is set by carry from bit 15.

Example: ADD IX, SP

Before: After:



ADD IY, rr

Add IY and register pair rr.

Function: IY ← IY + rr

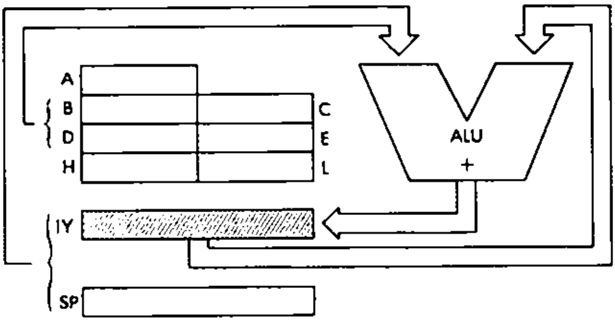
Format:

1	1	1	1	1	1	0	1	byte 1: FD
0	0	r	r	1	0	0	1	byte 2

Description: The contents of the IY register are added to the contents of the specified register pair and the result is stored back in IY. rr may be any one of:

BC – 00	IY – 10
DE – 01	SP – 11

Data Flow:



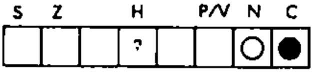
Timing: 4 M cycles; 15 T states: 7.5 usec @ 2 MHz

Addressing Mode: Implicit.

Byte Codes:

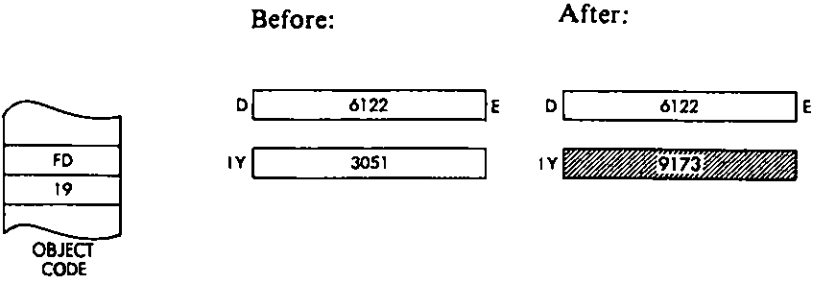
rr:	BC	DE	IY	SP
FD–	09	19	29	39

Flags:



H is set by carry out of bit 11.
C is set by carry out of bit 15.

Example: ADD IY, DE



AND s

Logical AND accumulator with operand s.

Function: A ← A ∧ s

Format: *s:* may be r, n, (HL), (IX + d), or (IY + d)

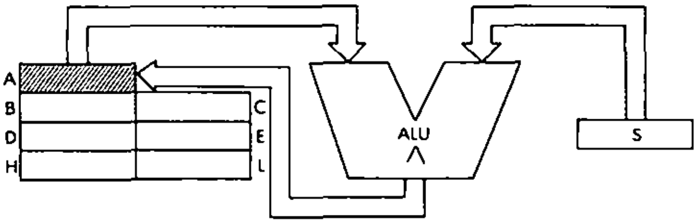
r	1	0	1	0	0	← r →			
n	1	1	1	0	0	1	1	0	byte 1: E6
	← n →								byte 2: immediate data
(HL)	1	0	1	0	0	1	1	0	A6
(IX + d)	1	1	0	1	1	1	0	1	byte 1: DD
	1	0	1	0	0	1	1	0	byte 2: A6
	← d →								byte 3: offset value
(IY + d)	1	1	1	1	1	1	0	1	byte 1: FD
	1	0	1	0	0	1	1	0	byte 2: A6
	← d →								byte 3: offset value

r may be any one of:

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

Description: The accumulator and the specified operand are logically 'and'ed and the result is stored in the accumulator. s is defined in the description of the similar ADD instructions.

Data Flow:



Timing:

s:	*M cycles:*	*T states:*	*usec @ 2 MHz:*
r	1	4	2
n	2	7	3.5
(HL)	2	7	3.5
(IX + d)	5	19	9.5
(IY + d)	5	19	9.5

Addressing Mode: r: implicit; n: immediate; (HL): indirect; (IX + d), (IY + d): indexed.

Byte Codes: AND r

r: A	B	C	D	E	H	L
A7	A0	A1	A2	A3	A4	A5

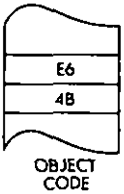
Flags:

S	Z		H		P/V	N	C
●	●		1		●	○	○

Example: AND 4B

Before: A 36

After: A 02



BIT b, (HL) Test bit b of indirectly addressed memory location (HL)

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

Format:

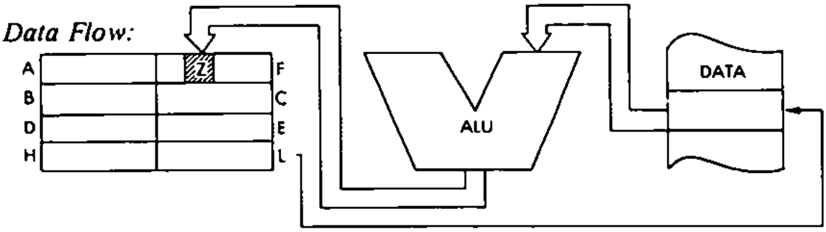
1	1	0	0	1	0	1	1	byte 1: CB·

0	1	←	b	→	1	1	0	byte 2

Description: The specified bit of the memory location addressed by the contents of the HL register pair is tested and the Z flag is set according to the result. b may be any one of:

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

Data Flow:



Timing: 3 M cycles; 12 T states; 6 usec @ 2 MHz

Addressing Mode: Indirect.

Flags:

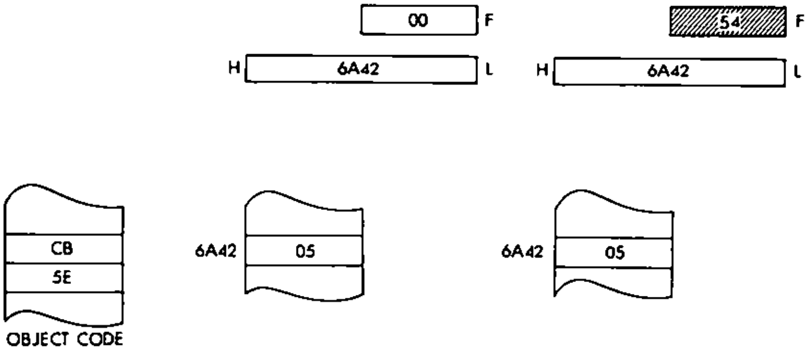
S	Z		H		P/V	N	C
?	●		1		?	0	

Byte Codes:

b:	0	1	2	3	4	5	6	7
CB-	46	4E	56	5E	66	6E	76	7E

Example: BIT 3, (HL)

Before: After:



BIT b, (IX + d) Test bit b of indexed addressed memory location (IX + d)

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

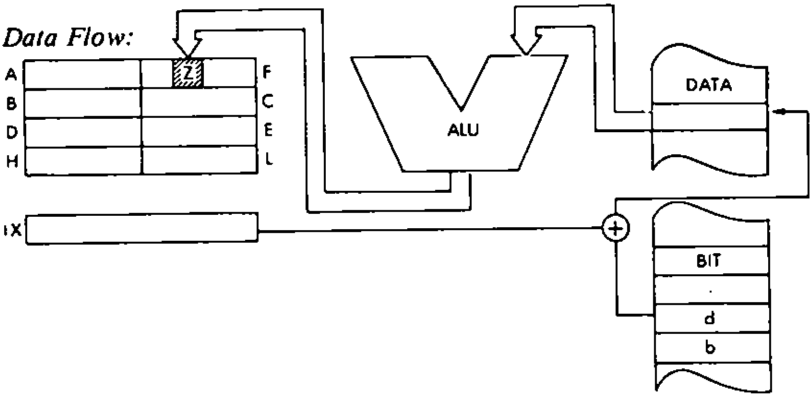
Format:

1	1	0	1	1	1	0	1	byte 1: DD
1	1	0	0	1	0	1	1	byte 2: CB
←			d				→	byte 3: offset value
0	1	←	b	→	1	1	0	byte 4

Description: The specified bit of the memory location addressed by the contents of the IX register plus the given offset value is tested and the Z flag is set according to the result. b may be any one of:

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

Data Flow:



Timing: 5 M cycles; 20 T states: 10 usec @ 2 MHz

Addressing Mode: Indexed.

Byte Codes:

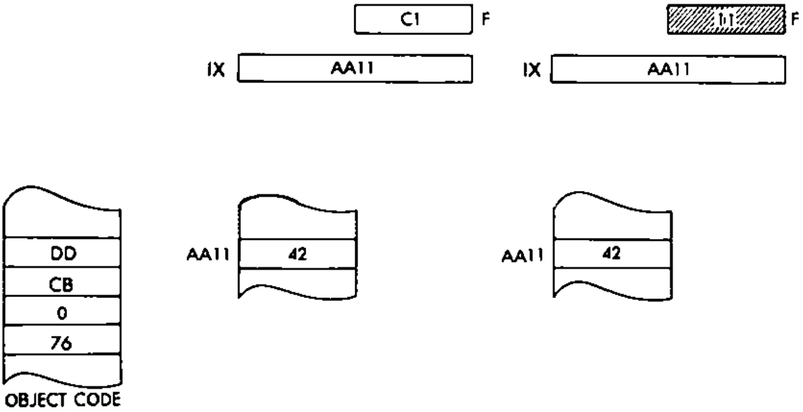
b:	0	1	2	3	4	5	6	7
DD-CB-d-	46	4E	56	5E	66	6E	76	7E

Flags:

S	Z		H		P/V	N	C
?	●		1		?	0	

Example: BIT 6, (IX + 0)

Before: After:



BIT b, (IY + d) Test bit b of the indexed addressed memory location (IY + d)

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

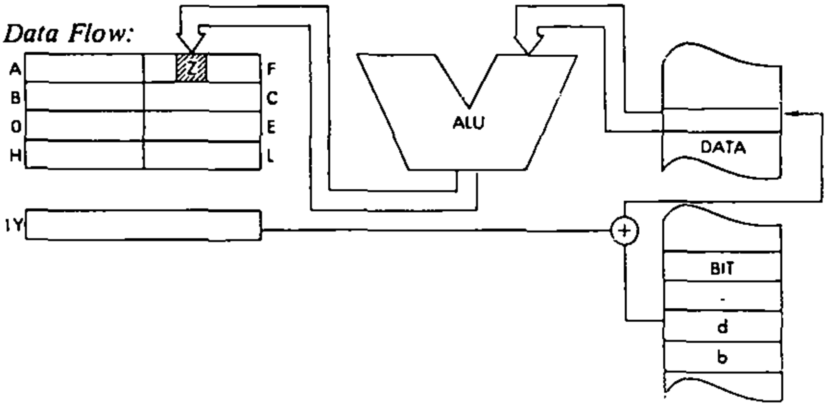
Format:

1	1	1	1	1	1	0	1	byte 1: FD
1	1	0	0	1	0	1	1	byte 2: CB
←			d				→	byte 3: offset value
0	1	←	b	→	1	1	0	byte 4

Description: The specified bit of the memory location addressed by the contents of the IY register plus the given offset value is tested and the Z flag is set according to the result. b may be any one of:

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

Data Flow:



Timing: 5 M cycles; 20 T states; 10 usec @ 2 MHz

Addressing Mode: Indexed.

Byte Codes: b::

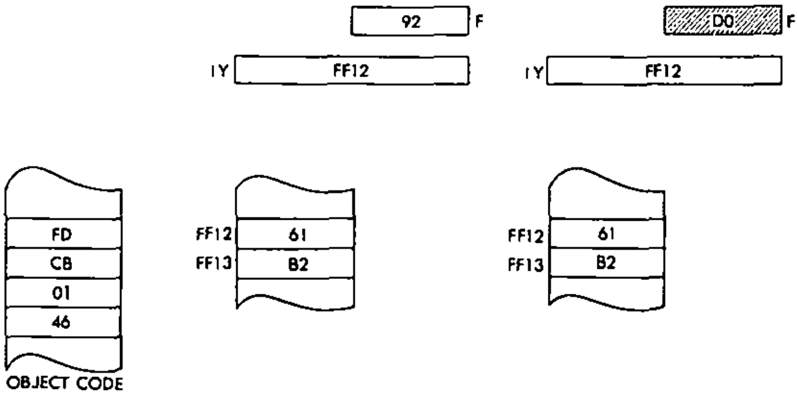
	0	1	2	3	4	5	6	7
FD-CB-d-	46	4E	56	5E	66	6E	76	7E

Flags:

S	Z		H		P/V	N	C
?	●		1		?	0	

Example: BIT 0, (IY + 1)

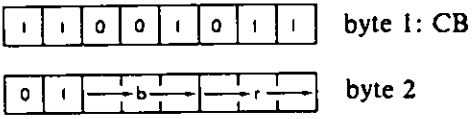
Before: After:



BIT b, r Test bit b of register r.

Function: $Z \leftarrow \overline{r_b}$

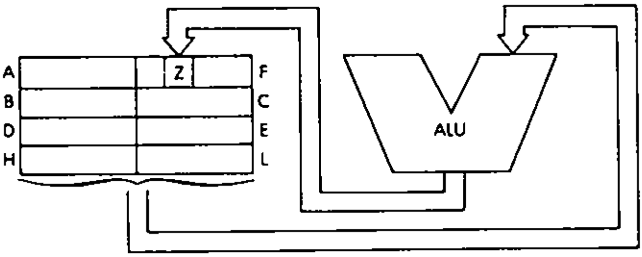
Format:



Description: The specified bit of the given register is tested and the zero flag is set according to the results. b and r may be any one of:

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

Data Flow:



Timing: 2 M cycles; 8 T states; 4 usec @ 2 MHz

Addressing Mode: Implicit.

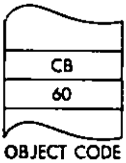
Byte Codes:

	b:	r: A	B	C	D	E	H	L
CB-	0	47	40	41	42	43	44	45
	1	4F	48	49	4A	4B	4C	4D
	2	57	50	51	52	53	54	55
	3	5F	58	59	5A	5B	5C	5D
	4	67	60	61	62	63	64	65
	5	6F	68	69	6A	6B	6C	6D
	6	77	70	71	72	73	74	75
	7	7F	7B	79	7A	7B	7C	7D

Flags:

S	Z		H		P/V	N	C
?	●		1		?	0	

Example: **BIT 4, B**



Before:

B 61 | 01 F

After:

B 61 | 55 F

CALL cc, pq

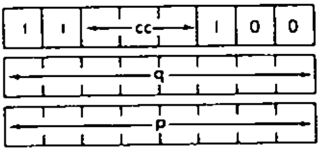
Call subroutine on condition.

Function:

if cc true: $(SP - 1) \leftarrow PC_{high}$; $(SP - 2) \leftarrow PC_{low}$; $SP \leftarrow SP - 2$; $PC \leftarrow pq$

If cc false: $PC \leftarrow PC + 3$

Format:



byte 1

byte 2: address, low order

byte 3: address, high order

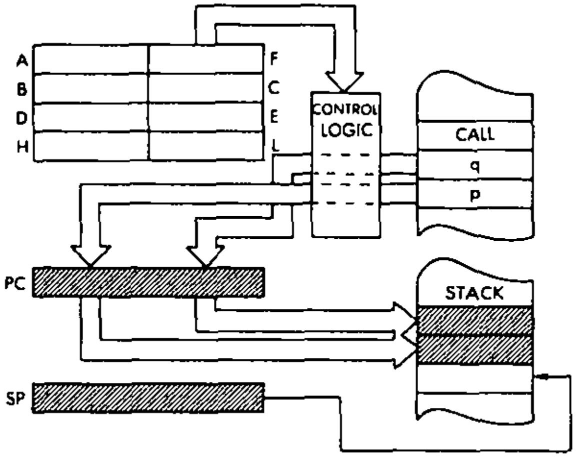
Description:

If the condition is met, the contents of the program counter are pushed onto the stack as described for the PUSH instructions. Then, the contents of the memory location immediately following the opcode are loaded into the low order of the PC and the contents of the second memory location after the the opcode are loaded into the high order half of the PC. The next instruction fetched will be from this new address. If the condition is not met, the address pq is ignored and the following instruction is executed. cc may be any one of:

NZ	000	PO	100
Z	001	PE	101
NC	010	P	100
C	011	M	111

An RET instruction can be used at the end of the subroutine being called to restore the PC.

Data Flow:



Timing:

	M cycles:	*T states:*	*usec @ 2 MHz*
condition true:	5	17	8.5
condition not true:	3	10	5

Addressing Mode: Immediate.

Byte Codes:

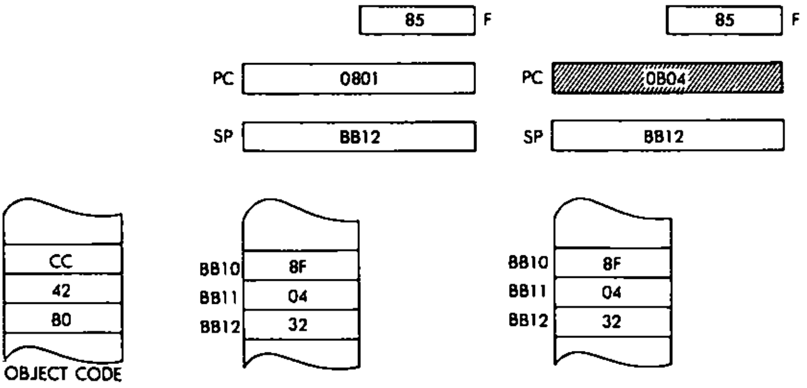
CC. NZ	Z	NC	C	PO	PE	P	M	
C4	CC	D4	DC	E4	EC	F4	FC	-q-p

Flags:

S	Z		H		P/V	N	C	
								(no effect)

Example: CALL Z, B042

Before: After:

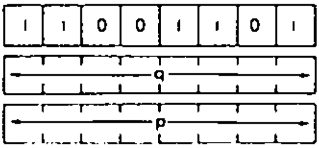


CALL pq

Call subroutine at location pq.

Function: (SP − 1) ← PC_{high}; (SP − 2) ← PC_{low}; SP ← SP − 2; PC ← pq

Format:



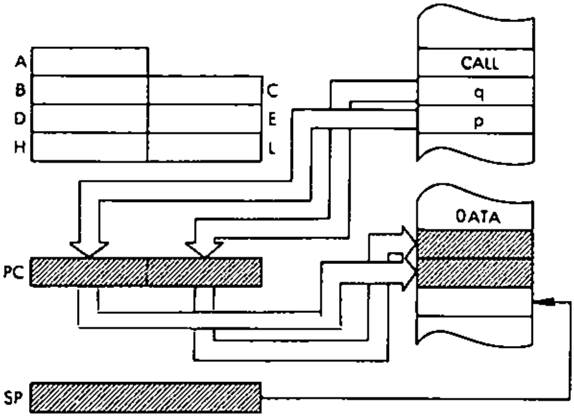
byte 1: CD 205

byte 2: address, low order

byte 3: address, high order

Description: The contents of the program counter are pushed onto the stack as described for the PUSH instructions. The contents of the memory location immediately following the opcode are then loaded into the low order half of the PC and the contents of the second memory location after the opcode are loaded in the high order half of the PC. The next instruction will be fetched from this new address.

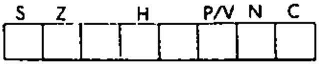
Data Flow:



Timing: 5 M cycles; 17 T states: 8.5 usec @ 2 MHz

Addressing Mode: Immediate.

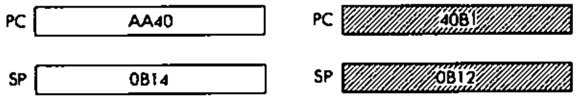
Flags:

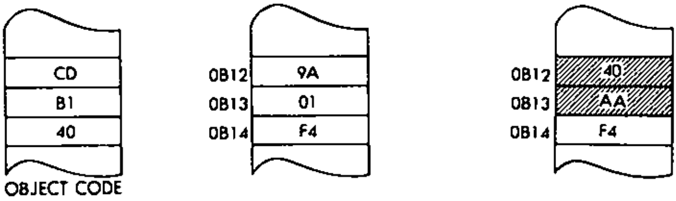


(no effect)

Example: CALL 40B1

Before: After:





CCF

Complement carry flag.

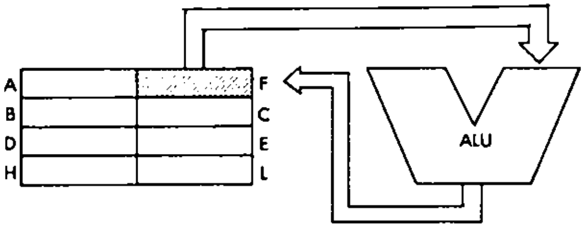
Function: $C \leftarrow \overline{C}$

Format:

0	0	1	1	1	1	1	1	3F

Description: The carry flag is complemented.

Data Flow:



Timing: 1 M cycle; 4 T states: 2 usec @ 2 MHz

Addressing Mode: Implicit.

Flags:

S	Z		H		P/V	N	C
			?			○	●

CP s

Compare operand s to accumulator.

Function: A − s

Format: s: may be r, n, (HL), (IX + d), or (IY + d).

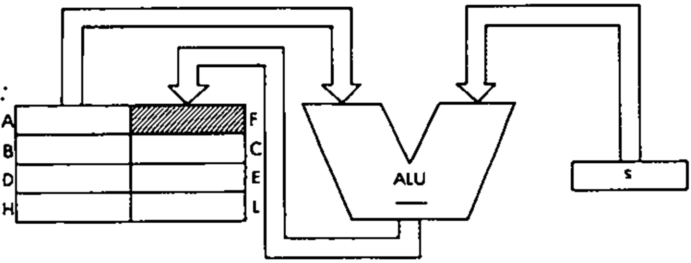
r	1	0	1	1	1	←r→			
n	1	1	1	1	1	1	1	0	FE
	←n→								byte 2: immediate data
(HL)	1	0	1	1	1	1	1	0	byte 1: BE
(IX + d)	1	1	0	1	1	1	0	1	byte 1: DD
	1	0	1	1	1	1	1	0	byte 2: BE
	←d→								byte 3: offset value
(IY + d)	1	1	1	1	1	1	0	1	byte 1: FD
	1	0	1	1	1	1	1	0	byte 2: BE
	←d→								byte 3: offset value

r may be any one of:

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

Description: The specified operand is subtracted from the accumulator, and the result is discarded. s is defined in the description of the similar ADD instructions.

Data Flow:



Timing:

s:	*M cycles:*	*T states:*	*usec @ 2 MHz:*
r	1	4	2
n	2	7	3.5
(HL)	2	7	3.5
(IX + d)	5	19	9.5
(IY + d)	5	19	9.5

Addressing Modes: r: implicit; n: immediate; (HL): indirect; (IX + d), (IY + d): indexed

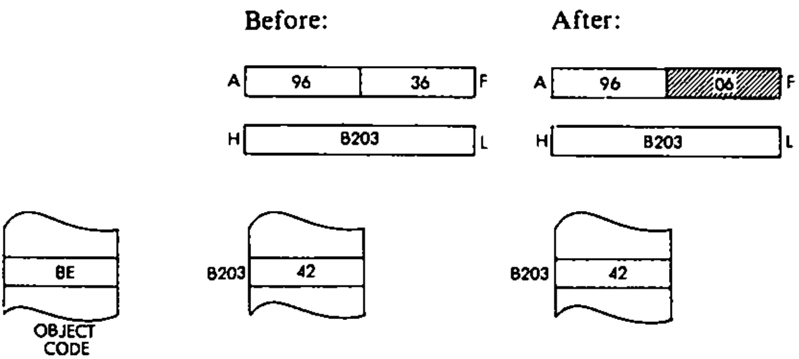
Byte Codes: CP r:

r:	A	B	C	D	E	H	L
	BF	B8	B9	BA	BB	BC	BD

Flags:

S	Z		H		P/V	N	C
●	●		●		●	1	●

Example: CP (HL)



CPD

Compare with decrement.

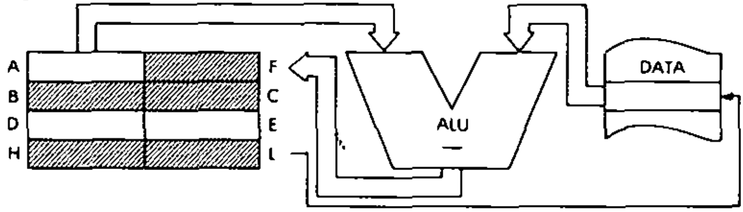
Function: A — [HL]; HL ← HL — 1; BC ← BC — 1

Format:

1	1	1	0	1	1	0	1	byte 1: ED
1	0	1	0	1	0	0	1	byte 2: A9

Description: The contents of the memory location addressed by the HL register pair are subtracted from the contents of the accumulator and the result is discarded. Then both the HL register pair and the BC register pair are decremented.

Data Flow:



Timing: 4 M cycles; 16 T states: 8 usec @ 2 MHz

Addressing Mode: indirect.

Flags:

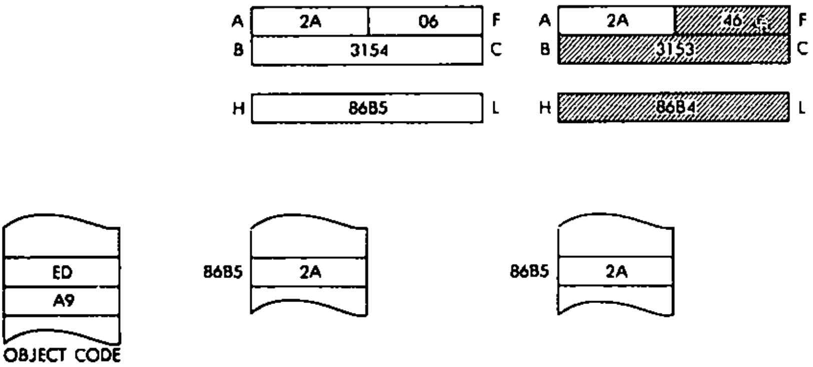
S	Z		H		P/V	N	C
●	X		●		X	1	

P/V: Reset if BC = 0 after execution; set otherwise

Z: Set if A = [HL]

Example: CPD

Before: After:



CPDR

Block compare with decrement.

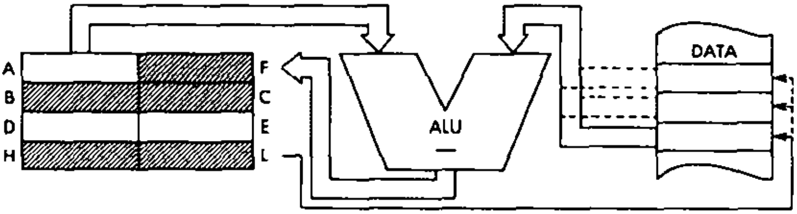
Function: A — [HL]; HL ← HL — 1; BC ← BC — 1;
Repeat until BC = 0 or A = [HL]

Format:

1	1	1	0	1	1	0	1	byte 1: ED
1	0	1	1	1	0	0	1	byte 2: B9

Description: The contents of the memory location addressed by the HL register pair are subtracted from the contents of the accumulator and the result is discarded. Then both the BC register pair and the HL register pair are decremented. If BC ≠ 0 and A ≠ [HL], the program counter is decremented by two and the instruction is re-executed.

Data Flow:



Timing: BC = 0 or A = [HL]: 4 M cycles; 16 T states: 8 usec @ 2 MHz
BC ≠ 0 and A ≠ [HL]: 5 M cycles; 21 T states: 10.5 usec @ 2 MHz

Flags:

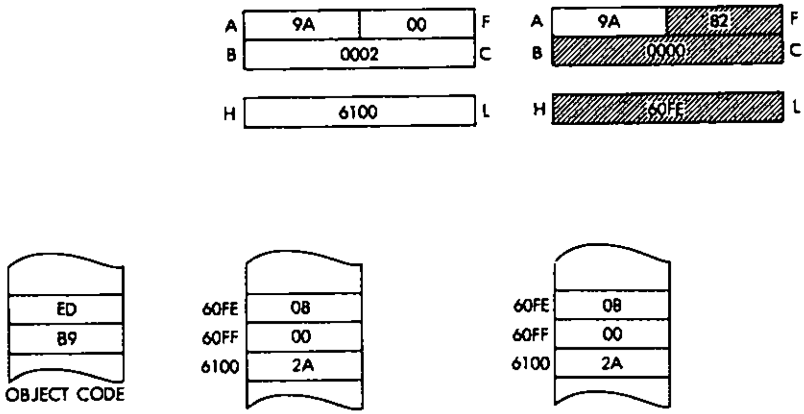
S	Z		H		P/V	N	C
●	x		●		x	1	

Reset if BC = 0 after execution; set otherwise

Set if A = [HL]

Example: CPDR

Before: After:



CPI

Compare with increment.

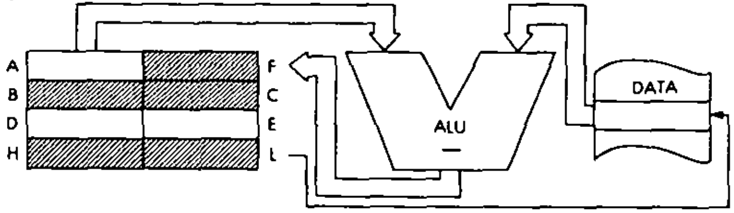
Function: A — [HL]; HL ← HL + 1; BC ← BC — 1

Format:

1	1	1	0	1	1	0	1	byte 1: ED
1	0	1	0	0	0	0	1	byte 2: A1

Description: The contents of the memory location addressed by the HL register pair are subtracted from the contents of the accumulator and the result is discarded. The HL register pair is incremented and the BC register pair is decremented.

Data Flow:



Timing: 4 M cycles; 16 T states: 8 usec @ 2 MHz

Addressing Mode: indirect.

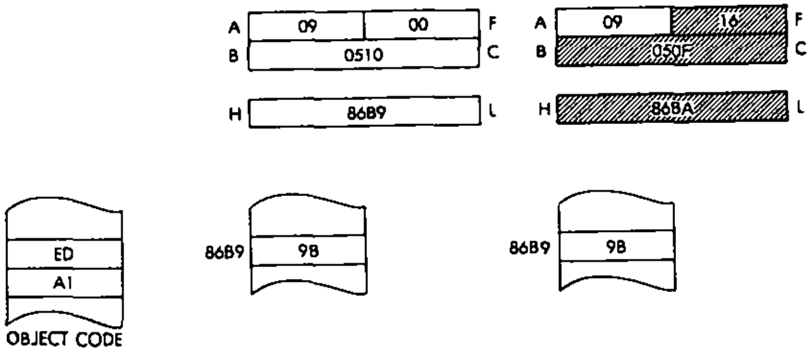
Flags:

S	Z		H		P/V	N	C
●	X		●		X	1	

Reset if BC = 0 after execution set otherwise
Set if A = [HL]

Example: CPI

Before: After:



CPIR

Block compare with increment.

Function:

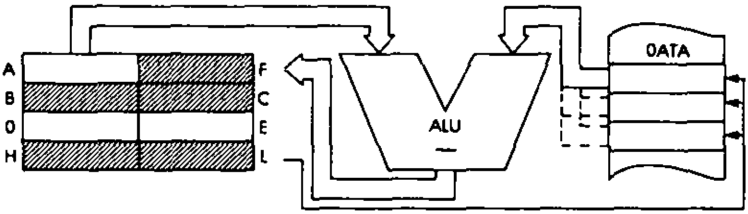
A — [HL]; HL ← HL + 1; BC ← BC — 1;
Repeat until BC = 0 or A = [HL]

1	1	1	0	1	1	0	1	byte 1: ED
1	0	1	1	0	0	0	1	byte 2: B1

Description:

The contents of the memory location addressed by the HL register pair are subtracted from the contents of the accumulator and the result is discarded. Then the HL register pair is incremented and the BC register pair is decremented. If BC ≠ 0 and A ≠ [HL], then the program counter is decremented by 2 and the instruction is re-executed.

Data Flow:

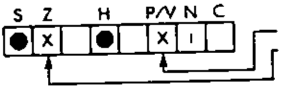


Timing:

BC = 0 or A = [HL] : 4 M cycles; 16 T states: 8 usec @ 2 MHz
BC ≠ 0 and A ≠ [HL] : 5 M cycles; 21 T states: 10.5 usec @ 2 MHz

Addressing Mode: indirect.

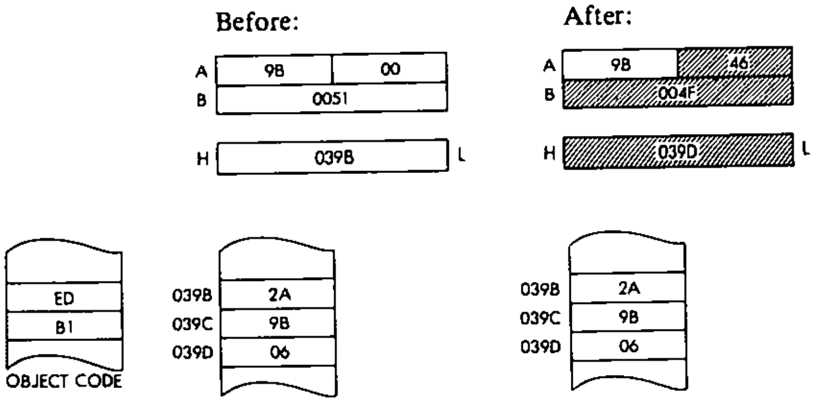
Flags:



Reset if BC = 0 after execution; set otherwise

Set if A = [HL]

Example: CPIR



CPL

Complement accumulator.

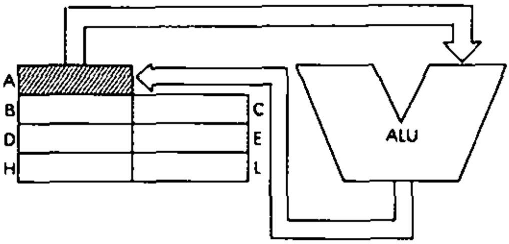
Function: $A \leftarrow \overline{A}$

Format:

0	0	1	0	1	1	1	1	2F

Description: The contents of the accumulator are complemented, or inverted, and the result is stored back in the accumulator (one's complement).

Data Flow:



Timing: 1 M cycle; 4 T states; 2 usec @ 2 MHz

Addressing Mode: Implicit.

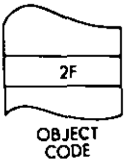
Flags:

S	Z		H		P/V	N	C
			1			1	

Example: CPL

Before: A 3D

After: A C2



DAA

Decimal adjust accumulator.

Function: See below.

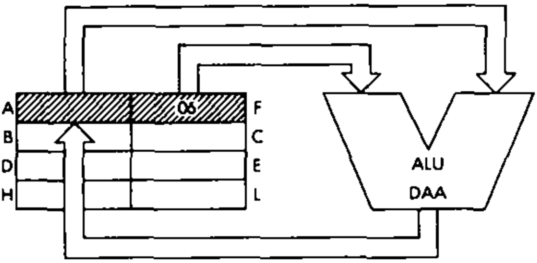
Format:

0	0	1	0	0	1	1	1	27

Description: The instruction conditionally adds "6" to the right and/or left nibble of the accumulator, based on the status register, for BCD conversion after arithmetic operations.

N	C	*value of* high nibble	H	*value of* low nibble	# added to A	C after execution
0 (ADD, ADC, INC)	0	0-9	0	0-9	00	0
	0	0-8	0	A-F	06	0
	0	0-9	1	0-3	06	0
	0	A-F	0	0-9	60	1
	0	9-F	0	A-F	66	1
	0	A-F	1	0-3	66	1
	1	0-2	0	0-9	60	1
	1	0-2	0	A-F	66	1
	1	0-3	1	0-3	66	1
1 (SUB, SBC, DEC, NEG)	0	0-9	0	0-9	00	0
	0	0-8	1	6-F	FA	0
	1	7-F	0	0-9	AO	1
	1	6-F	1	6-F	9A	1

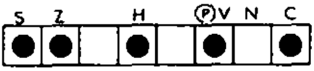
Data Flow:



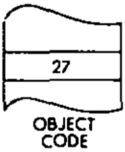
Timing: 1 M cycle; 4 T states; 2 usec @ 2 MHz

Addressing Mode: Implicit.

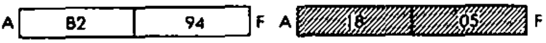
Flags:



Example: DAA



Before: After:



DEC m

Decrement operand m.

Function: m ← m − 1

Format: m: may be r, (HL), (IX+d), (IY+d)

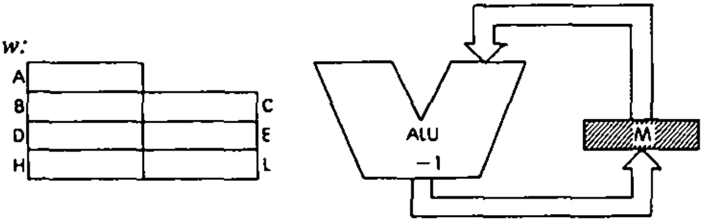
r	0	0	← r →			1	0	1	
(HL)	0	0	1	1	0	1	0	1	35
(IX + d)	1	1	0	1	1	1	0	1	byte 1: DD
	0	0	1	1	0	1	0	1	byte 2: 35
	← d →								byte 3: offset value
(IY + d)	1	1	1	1	1	1	0	1	byte 1: FD
	0	0	1	1	0	1	0	1	byte 2: 35
	← d →								byte 3: offset value

r may be any one of:

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

Description: The contents of the location addressed by the specific operand are decremented and stored back at that location. m is defined in the description of the similar INC instructions.

Data Flow:



Timing:

m:	*M cycles:*	*T states:*	*usec @ 2 MHz:*
r	1	4	2
(HL)	3	11	5.5
(IX + d)	6	23	11.5
(IY + d)	6	23	11.5

Addressing Mode: r: implicit; (HL): indirect; (IX + d), (IY + d): indexed.

Byte Codes: DEC r

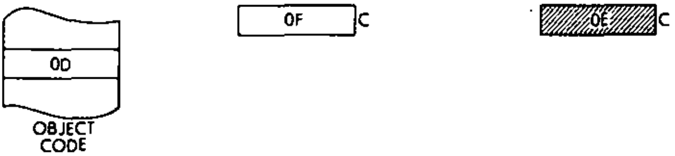
r:	A	B	C	D	E	H	L
	3D	05	0D	15	1D	25	2D

Flags:

S	Z		H		P/Ⓥ	N	C
●	●		●		●	1	

Example: DEC C

Before: After:



DEC rr

Decrement register pair rr.

Function: rr ← rr − 1

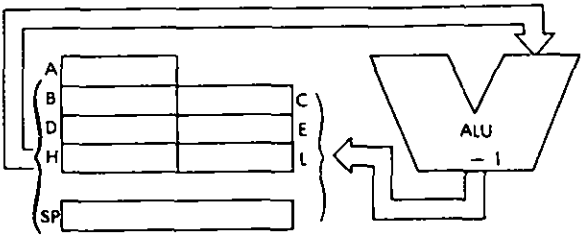
Format:

0	0	r	r	1	0	1	1

Description: The contents of the specified register pair are decremented and the result is stored back in the register pair. rr may be any one of:

BC − 00	HL − 10
DE − 01	SP − 11

Data Flow:



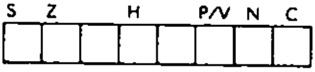
Timing: 1 M cycle; 6 T states; 3 usec @ 2 MHz

Addressing Mode: Implicit.

Byte Codes: rr:

BC	DE	HL	SP
0B	1B	2B	3B

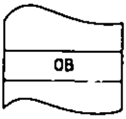
Flags:



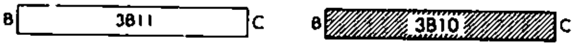
(no effect).

Example: DEC BC

Before: After:



OBJECT CODE



DEC IX

Decrement IX.

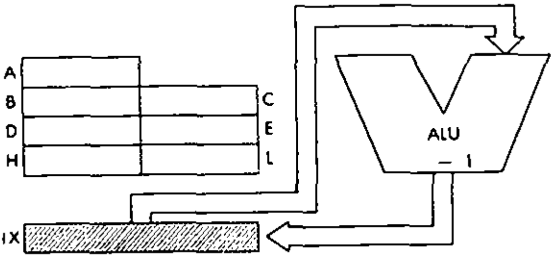
Function: IX ← IX − 1

Format:

1	1	0	1	1	1	0	1	byte 1: DD
0	0	1	0	1	0	1	1	byte 2: 2B

Description: The contents of the IX register are decremented and the result is stored back in IX.

Data Flow:



Timing: 2 M cycles; 10 T states; 5 usec @ 2 MHz

Addressing Modes: Implicit.

Flags:

S	Z		H		P/V	N	C

(no effect).

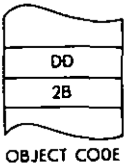
Example: DEC IX

Before:

IX 6114

After:

IX 6113



OBJECT CODE

DEC IY

Decrement IY.

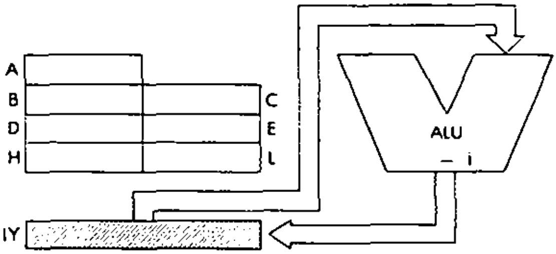
Function: IY ← IY − 1

Format:

1	1	1	1	1	1	0	1	byte 1: FD
0	0	1	0	1	0	1	1	byte 2: 2B

Description: The contents of the IY register are decrementea and the result is stored back in IY.

Data Flow:



Timing: 2 M cycles; 10 T states; 5 usec @ 2 MHz

Addressing Mode: Implicit.

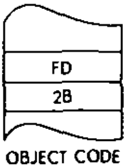
Flags:

S	Z		H		P/V	N	C	
								(no effect).

Example: DEC IY

Before: IY 900F

After: IY 900E



DI

Disable interrupts.

Function: IFF ← 0

Format:

1	1	1	1	0	0	1	1	F3

Description: The interrupt flip-flops are reset, thereby disabling all maskable interrupts. It is reenabled by an EI instruction.

Timing: 1 M cycle; 4 T states; 2 usec @ 2 MHz

Addressing Mode: Implicit.

Flags:

S	Z		H		P/V	N	C	
								(no effect).

DJNZ e

Decrement B and jump e relative on no zero.

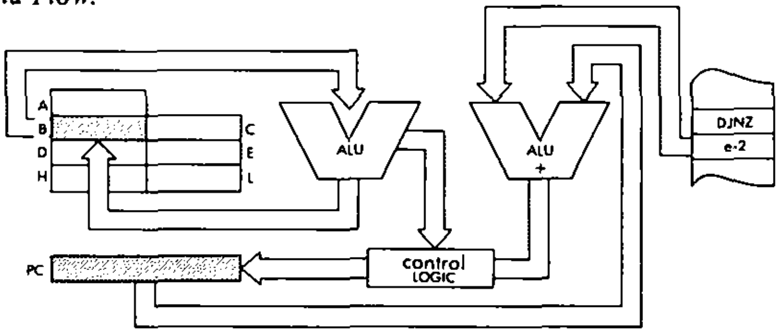
Function: $B \leftarrow B - 1$; if $B \neq 0$: $PC \leftarrow PC + e$

Format:

0	0	0	1	0	0	0	0	byte 1: 10
e-2								byte 2: offset value

Description: The B register is decremented. If the result is not zero, the immediate offset value is added to the program counter using two's complement arithmetic so as to enable both forward and backward jumps. The offset value is added to the value of PC + 2 (after the jump). As a result, the effective offset is -126 to +129 bytes. The assembler automatically subtracts from the source offset value to generate the hex code.

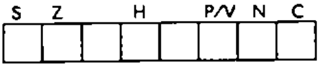
Data Flow:



Timing: B ≠ 0: 3 M cycles; 13 T states; 6.5 usec @ 2 MHz.
B = 0: 2 M cycles; 8 T states; 4 usec @ 2 MHz

Addressing Modes: Immediate.

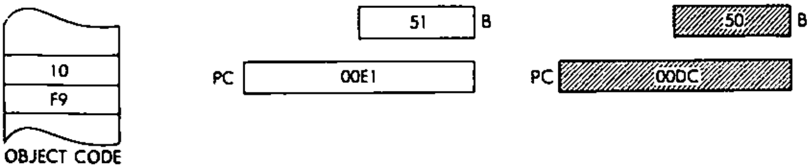
Flags:



(no effect)

Example: DJNZ $ - 5 ($ = current PC)

Before: After:



EI

Enable interrupts.

Function: IFF ← 1

Format:

1	1	1	1	1	0	1	1	FB

Description: The interrupt flip-flops are set, thereby enabling maskable interrupts after the execution of the instruction following the EI instruction. In the meantime maskable interrupts are disabled.

Timing: 1 M cycle; 4 T states; 2 usec @ 2 MHz

Addressing Mode: Implicit.

Flags:

S	Z		H		P/V	N	C	
								(no effect).

Example: A usual sequence at the end of an interrupt routine is:

EI
RETI

The maskable interrupt is re-enabled following completion of RETI.

EX AF, AF'

Exchange accumulator and flags with alternate registers.

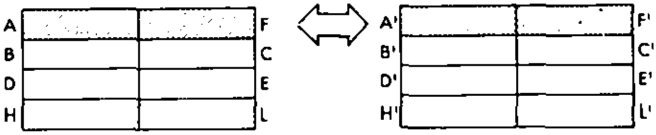
Function: AF↔AF'

Format:

0	0	0	0	1	0	0	0	08

Description: The contents of the accumulator and status register are exchanged with the contents of the alternate accumulator and status register.

Data Flow:



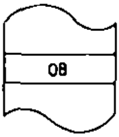
Timing: 1 M cycle; 4 T states; 2 usec @ 2 MHz

Addressing Mode: Implicit.

Flags:

S	Z		H		P/V	N	C
●	●	●	●	●	●	●	●

Example: EX AF, AF'



OBJECT CODE

Before:

A	04	81	F
A'	90	3A	F'

After:

A	90	3A	F
A'	04	81	F'

EX DE, HL

Exchange the HL and DE registers.

Function: DE ←→ HL

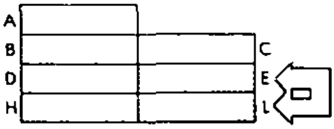
Format:

1	1	1	0	1	0	1	1

EB

Description: The contents of the register pairs DE and HL are exchanged.

Data Flow:



Timing: 1 M cycle; 4 T states; 2 usec @ 2 MHz

Addressing Mode: Implicit.

Flags:

S	Z		H		P/V	N	C

(no effect).

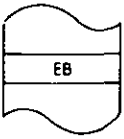
Example: EX DE, HL

Before:

D	A4E6	E
H	9604	L

After:

D	9604	E
H	A4E6	L



OBJECT CODE

EX (SP), HL

Exchange HL with top of stack.

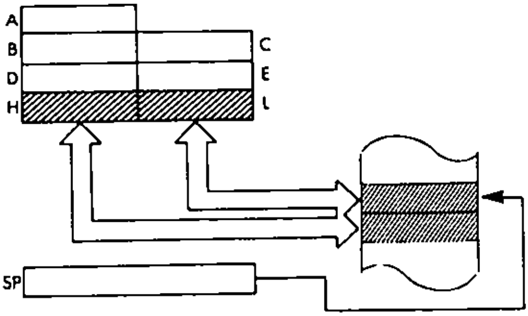
Function: (SP) ←L; (SP + 1) ← H

Format:

1	1	1	0	0	0	1	1	E3

Description: The contents of the L register are exchanged with the contents of the memory location addressed by the stack pointer. The contents of the H register are exchanged with the contents of the memory location immediately following the one addressed by the stack pointer.

Data Flow:



Timing: 5 M cycles; 19 T states; 9.5 usec @ 2 MHz

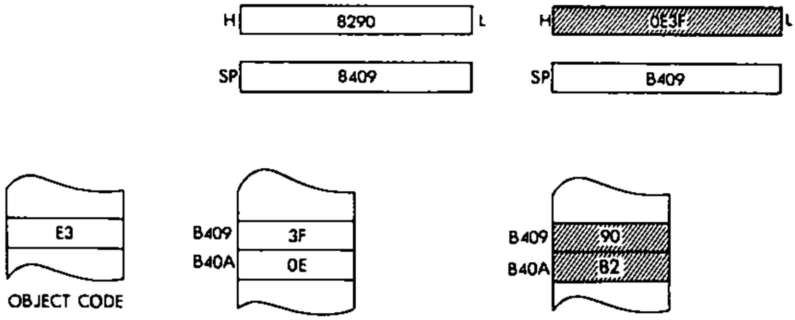
Addressing Mode: Indirect.

Flags:

S	Z		H		P/V	N	C	
								(no effect).

Example: EX (SP), HL

Before: After



EX (SP), IX

Exchange IX with top of stack.

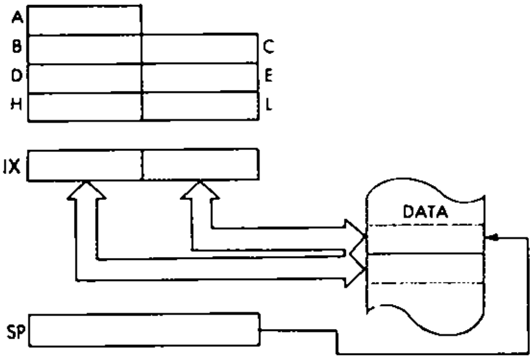
Function: $(SP) \leftrightarrow IX_{low}$; $(SP + 1) \leftrightarrow IX_{high}$

Format:

1	1	0	1	1	1	0	1	byte 1: DD
1	1	1	0	0	0	1	1	byte 2: E3

Description: The contents of the low order of the IX register are exchanged with the contents of the memory location addressed by the stack pointer. The contents of the high order of the IX register are exchanged with the contents of the memory location immediately following the one addressed by the stack pointer.

Data Flow:



Timing: 6 M cycles; 23 T states; 11.5 usec @ 2 MHz

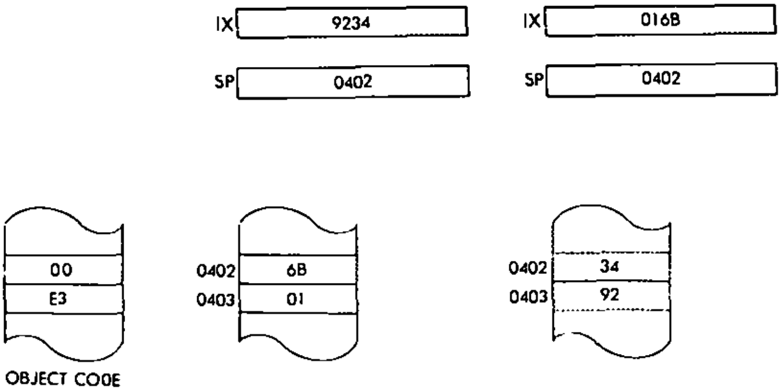
Addressing Mode: Indirect.

Flags:

S	Z		H		P/V	N	C	
								(no effect).

Example: EX (SP), IX

Before: After:



EX (SP), IY Exchange IY with top of stack.

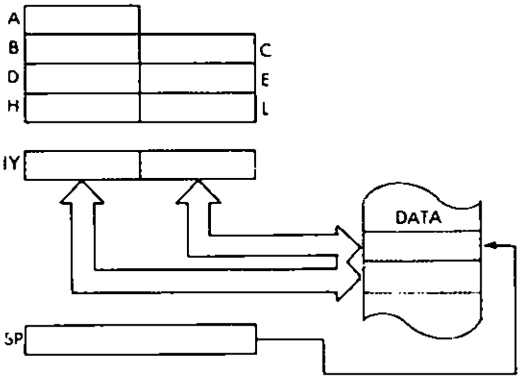
Function: $(SP) \leftrightarrow IY_{low}; (SP + 1) \leftrightarrow IY_{high}$

Format:

1	1	1	1	1	1	0	1	byte 1: FD
1	1	1	0	0	0	1	1	byte 2: E3

Description: The contents of the low order of the IY register are exchanged with the contents of the memory location addressed by the stack pointer. The contents of the high order of the IY register are exchanged with the contents of the memory location immediately following the one addressed by the stack pointer.

Data Flow:



Timing: 6 M cycles; 23 T states; 11.5 usec @ 2 MHz

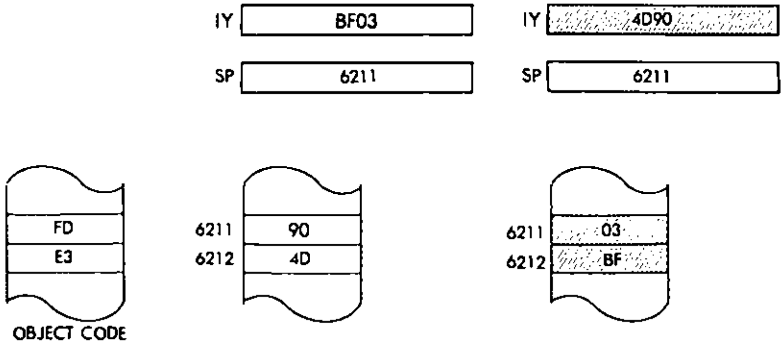
Addressing Mode: Indirect.

Flags:

S	Z		H		P/V	N	C	
								(no effect).

Example: EX (SP), IY

Before: After:



EXX

Exchange alternate registers.

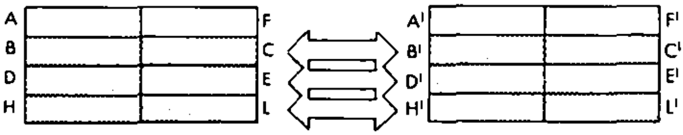
Function: BC ←→ BC'; DE ←→ DE'; HL ←→ HL'

Format:

1	1	0	1	1	0	0	1	D9

Description: The contents of the general purpose registers are exchanged with the contents of the corresponding alternate registers.

Data Flow:



Timing: 1 M cycle; 4 T states; 2 usec @ 2 MHz

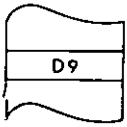
Addressing Mode: Implicit.

Flags:

S	Z		H		P/V	N	C	
								(no effect).

Example: EXX

	Before:				After:		
A	04	2B	F	A	04	2B	F
B	39	26	C	B	8C	00	C
D	54	02	E	D	93	D0	E
H	F1	D0	L	H	4F	E3	L
A'	3F	2A	F'	A'	3F	2A	F'
B'	8C	00	C'	B'	39	26	C'
D'	93	D0	E'	D'	54	02	E'
H'	4F	E3	L'	H'	F1	D0	L'



OBJECT CODE

HALT *Halt CPU.*

Function: CPU suspended.

Format:

0	1	1	1	0	1	1	0	76

Description: CPU suspends operation and executes NOP's so as to continue memory refresh cycles, until interrupt or reset is received.

Timing: 1 M cycle; 4 T states; 2 usec @ 2 MHz + indefinite Nop's.

Addressing Mode: Implicit.

Flags:

S	Z		H		P/V	N	C	
								(no effect).

IM 0

Set interrupt mode 0 condition.

Function: Internal interrupt control.

Format:

1	1	1	0	1	1	0	1	byte 1: ED

0	1	0	0	0	1	1	0	byte 2: 46

Description: Sets interrupt mode 0. In this condition, the interrupting device may insert one instruction onto the data bus for execution, the first byte of which must occur during the interrupt acknowledge cycle.

Timing: 2 M cycle; 8 T states; 4 usec @ 2 MHz

Addressing Mode: Implicit.

Flags:

S	Z		H		P/V	N	C	
								(no effect).

IM 1

Set interrupt mode 1 condition.

Function: Internal interrupt control.

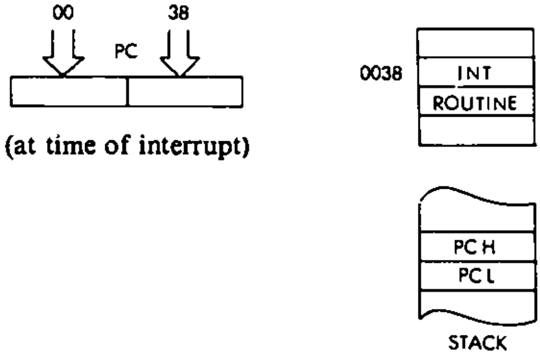
Format:

1	1	1	0	1	1	0	1	byte 1: ED

0	1	0	1	0	1	1	0	byte 2: 56

Description: Sets interrupt mode 1. A RST 0038H instruction will be executed when an interrupt occurs.

Data Flow:



(at time of interrupt)

Timing: 2 M cycles; 8 T states; 4 usec @ 2 MHz

Addressing Mode: Implicit.

Flags:

S	Z		H		P/V	N	C

(no effect).

IM 2

Set interrupt mode 2 condition.

Function: Internal interrupt control.

Format:

1	1	1	0	1	1	0	1	byte 1: ED

0	1	0	1	1	1	1	0	byte 2: 5E

Description: Set interrupt mode 2. When an interrupt occurs, one byte of data must be provided by the peripheral which is used as the low order of an address. The high order of this vector address is taken from the contents of the I register. This points to a second address stored in memory,which is loaded into the program counter and begins execution.

Timing: 2 M cycles; 8 T states; 4 usec @ 2 MHz

Addressing Mode: Implicit.

Flags:

S	Z		H		P/V	N	C	
								(no effect)

IN r, (C)

Load register r from port(C)

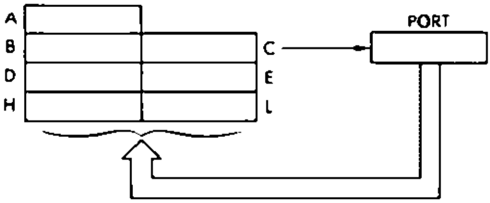
Function: r ← (C)

Format:

1	1	1	0	1	1	0	1	byte 1: ED
0	1	←	r	→	0	0	0	byte 2

Description: The peripheral device addressed by the contents of the C register is read and the result is loaded into the specified register.
C provides bits A0 to A7 of the address bus.
B provides bits A8 to A15.

Data Flow:



r may be any one of:

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

Timing: 3 M cycles; 12 T states; 6 usec @ 2 MHz

Addressing Mode: External.

Byte Codes:

r:	A	B	C	D	E	H	L
ED	78	40	48	50	58	60	68

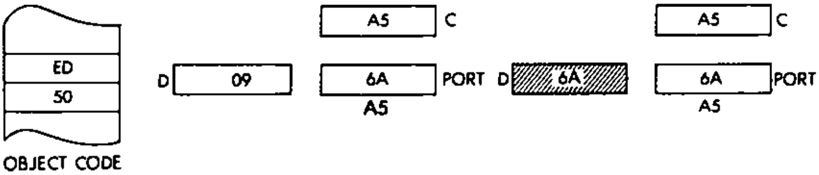
Flags:

S	Z		H		P/V	N	C
●	●		●		●	O	

It is important to note that INA,(N) does not have any effect on the flags, while IN r, (C) does.

Example: IN D, (C)

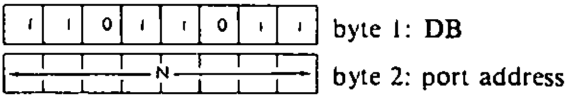
Before: After:



IN A, (N) Load accumulator from input port N.

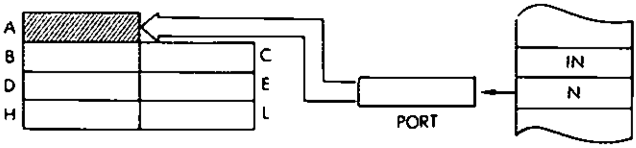
Function: A ← (N)

Format:



Description: The peripheral device N is read and the result is loaded into the accumulator.
The literal N is placed on lines A0 to A7 of the address bus. A supplies bits A8 to A15.

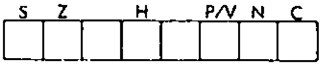
Data Flow:



Timing: 3 M cycles; 11 T states; 5.5 usec @ 2 MHz

Addressing Mode: External.

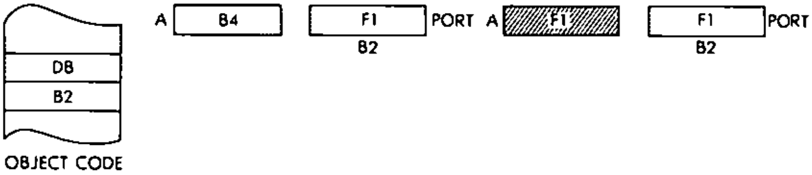
Flags:



(no effect).

Example: IN A, (B2)

Before: After:



INC r *Increment register r.*

Function: r ← r + 1

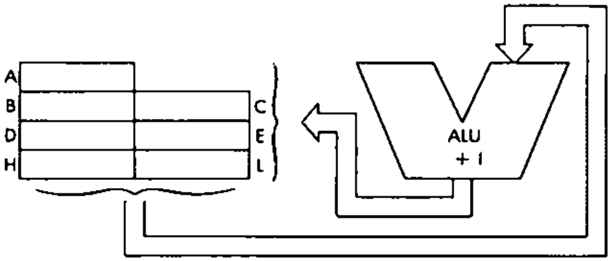
Format:

0	0	←	r	→	1	0	0

Description: The contents of the specified register are incremented. r may be any one of:

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

Data Flow:



Timing: 1 M cycle; 4 T states; 2 usec @ 2 MHz

Addressing Mode: Implicit.

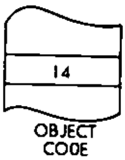
Byte Codes:

r:	A	B	C	D	E	H	L
	3C	04	0C	14	1C	24	2C

Flags:

S	Z		H		P/V	N	C
●	●		●		●	○	

Example: INC D



Before:

D 06

After:

D 07

INC rr

Increment register pair rr.

Function: rr ← rr + 1

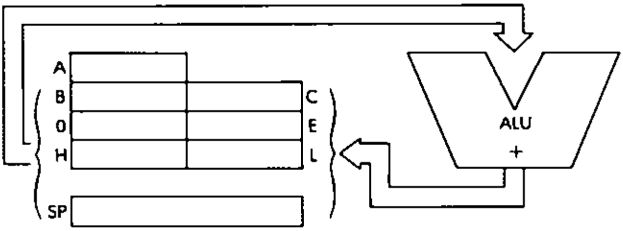
Format:

0	0	r	r	0	0	1	1

Description: The contents of the specified register pair are incremented and the result is stored back in the register pair. rr may be any one of:

BC – 00	HL – 10
DE – 01	SP – 11

Data Flow:



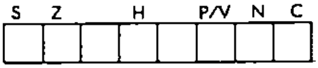
Timing: 1 M cycle; 6 T states; 3 usec @ 2 MHz

Addressing Mode: Implicit.

Byte Codes: rr:

BC	DE	HL	SP
03	13	23	33

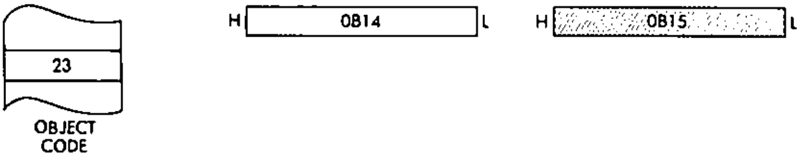
Flags:



(no effect).

Example: INC HL

Before: After:



INC (HL)

Increment indirectly addressed memory location (HL).

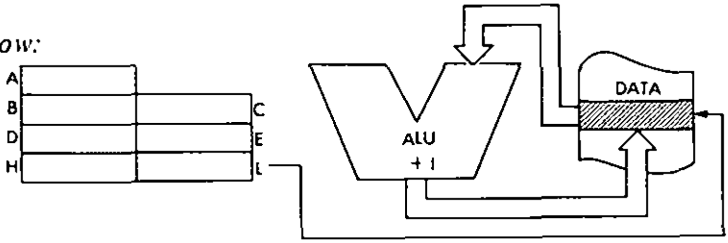
Function: (HL) ← (HL) + 1

Format:

0	0	1	1	0	1	0	0	34

Description: The contents of the memory location addressed by the HL register pair are incremented and stored back at that location.

Data Flow:



Timing: 3 M cycles; 11 T states; 5.5 usec @ 2 MHz

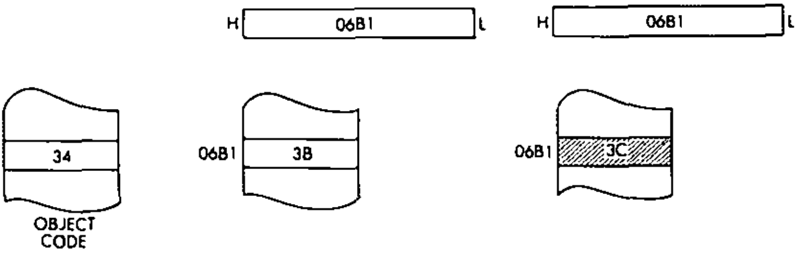
Addressing Mode: Indirect.

Flags:

S	Z		H		P/V	N	C
●	●		●		●	○	

Example: INC (HL)

Before: After:



INC (IX + d) Increment indexed addressed memory location (IX + d).

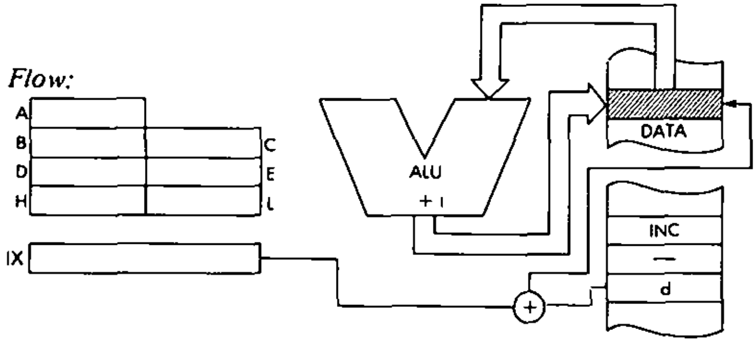
Function: (IX + d) ← (IX + d) + 1

Format:

1	1	0	1	1	1	0	1	byte 1: DD
0	0	1	1	0	1	0	0	byte 2: 34
←			d				→	byte 3: offset value

Description: The contents of the memory location addressed by the contents of the IX register plus the given offset value are incremented and stored back at that location.

Data Flow:



Timing: 6 M cycles; 23 T states; 11.5 usec @ 2 MHz

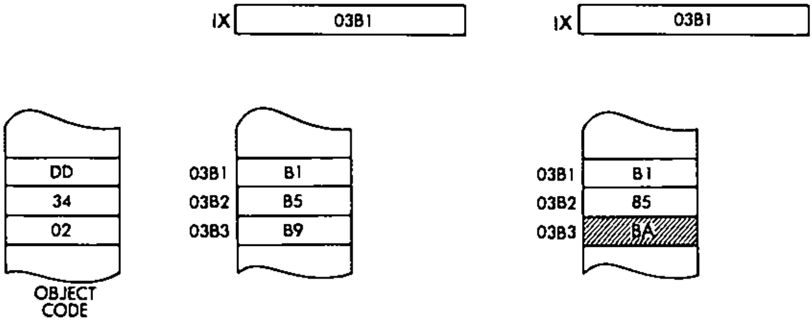
Addressing Mode: Indexed.

Flags:

S	Z		H		P/V	N	C
●	●		●		●	○	

Example: INC (IX + 2)

Before: After:



INC (IY + d)

Increment indexed addressed memory location (IY + d).

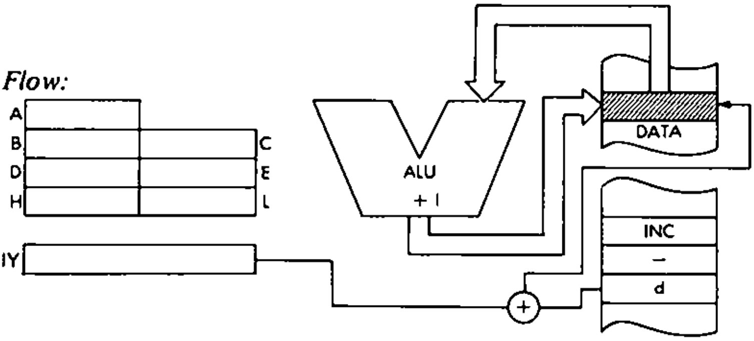
Function: (IY +d) ← (IY + d) + 1

Format:

1	1	1	1	1	1	0	1	byte 1: FD
0	0	1	1	0	1	0	0	byte 2: 34
←			d				→	byte 3: offset value

Description: The contents of the memory location addressed by the contents of the IY register plus the given offset value are incremented and stored back at that location.

Data Flow:



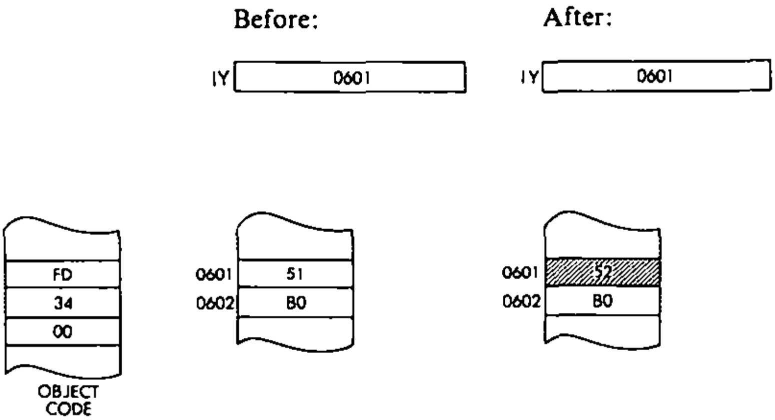
Timing: 6 M cycles; 23 T states; 11.5 usec @ 2 MHz

Addressing Mode: Indexed.

Flags:

S	Z		H		P/V	N	C
●	●		●		●	○	

Example: INC (IY + 0)



INC IX

Increment IX.

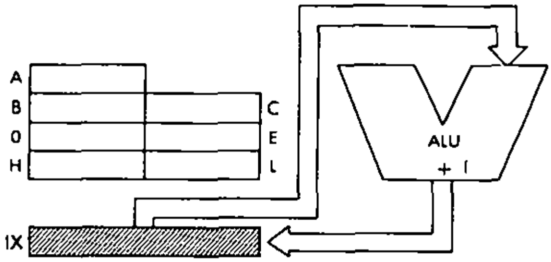
Function: IX ← IX + 1

Format:

1	1	0	1	1	1	0	1	byte 1: DD
0	0	1	0	0	0	1	1	byte 2: 23

Description: The contents of the IX register are incremented and the result is stored back in IX.

Data Flow:



Timing: 2 M cycles; 10 T states; 5 usec @ 2 MHz

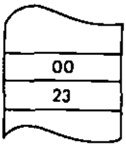
Addressing Mode: Implicit.

Flags:

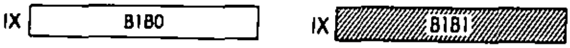
S	Z		H		P/V	N	C	
								(no effect).

Example: INC IX

Before: After:



OBJECT CODE



INC IY

Increment IY

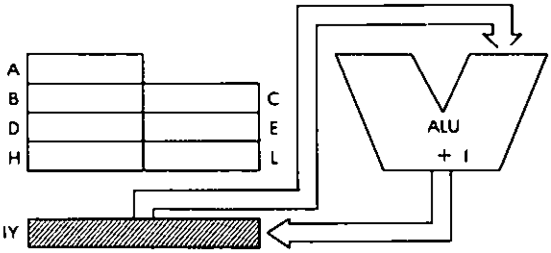
Function: IY ← IY + 1

Format:

1	1	1	1	1	1	0	1	byte 1: FD
0	0	1	0	0	0	1	1	byte 2: 23

Description: The contents of the IY register are incremented and the result is stored back in IY.

Data Flow:



Timing: 2 M cycles; 10 T states; 5 usec @ 2 MHz

Addressing Mode: Implicit.

Flags:

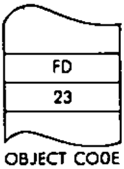
S	Z		H		P/V	N	C

(no effect).

Example: INC IY

Before: IY 36B1

After: IY 36B2



IND

Input with decrement.

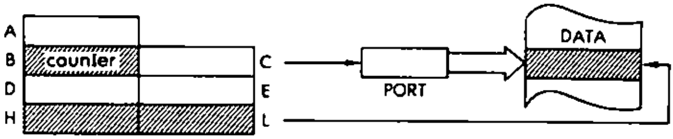
Function: (HL) ← (C); B ← B − 1; HL ← HL − 1

Format:

1	1	1	0	1	1	0	1	byte 1: ED
1	0	1	0	1	0	1	0	byte 2: AA

Description: The peripheral device addressed by the C register is read and the result is loaded into the memory location addressed by the HL register pair. The B register and the HL register pair are then each decremented.

Data Flow:



Timing: 4 M cycles; 16 T states; 8 usec @ 2 MHz

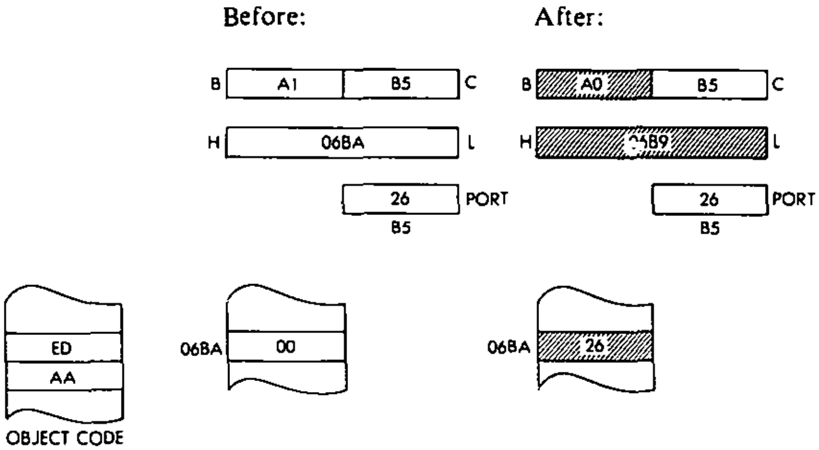
Addressing Mode: External.

Flags:

S	Z		H		P/V	N	C
?	X		?		?	1	

Z: Set if B = 0 after execution
Reset otherwise

Example: IND


Before:
After:
B
A1
B5
C
H
06BA
L
26
PORT
B5
B
A0
B5
C
H
06B9
L
26
PORT
B5
ED
AA
OBJECT CODE
06BA
00
06BA
26

INDR

Block input with decrement.

Function:

(HL) ← (C); B ← B − 1; HL ← HL − 1
Repeat until B = 0

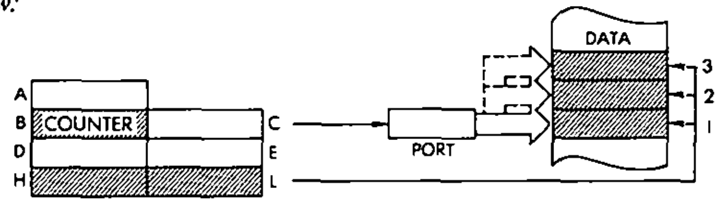
Format:

1	1	1	0	1	1	0	1	byte 1: ED
1	0	1	1	1	0	1	0	byte 2: BA

Description:

The peripheral device addressed by the C register is read and the result is loaded into the memory location addressed by the HL register pair. Then the B register and the HL register pair are decremented. If B is not zero, the program counter is decremented by 2 and the instruction is re-executed.

Data Flow:



Timing:

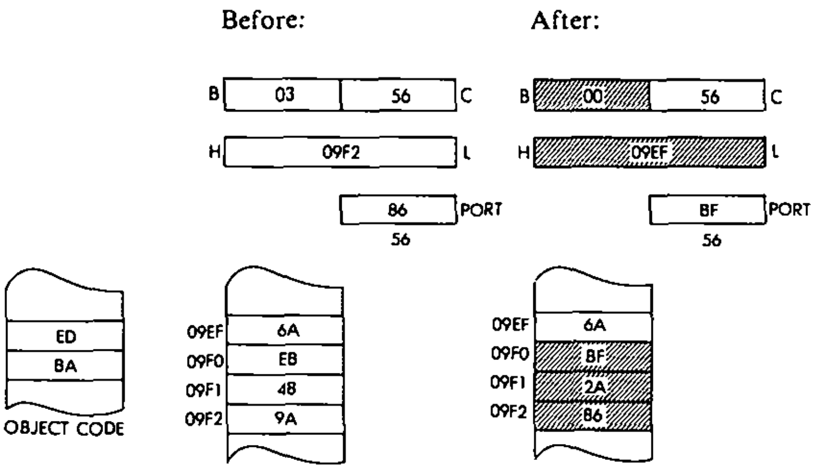
B = 0:4 M cycles; 16 T states; 8 usec @ 2 MHz.
B ≠ 0:5 M cycles; 21 T states; 10.5 usec @ 2 MHz.

Addressing Mode: External

Flags:

S	Z		H		P/V	N	C
?	1		?		?	1	

Example: INDR



INI

Input with increment.

Function: (HL) ← (C); B ← B − 1; HL ← HL + 1

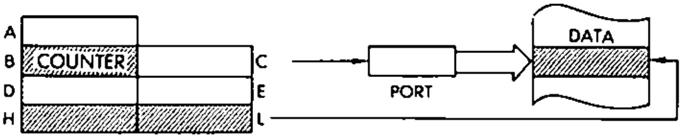
Format:

1	1	1	0	1	1	0	1	byte 1: ED
1	0	1	0	0	0	1	0	byte 2: A2

Description: The peripheral device addressed by the C register is read and the result is loaded into the memory location addressed by the HL register pair. The B register is decremented and the HL register pair is incremented.

The contents of C are placed on the low half of the address bus. The contents of B are placed on the high half. I/O selection is generally made by C, i.e., by A0 to A7. B is a byte counter.

Data Flow:



Timing: 4 M cycles; 16 T states; 8 usec @ 2 MHz

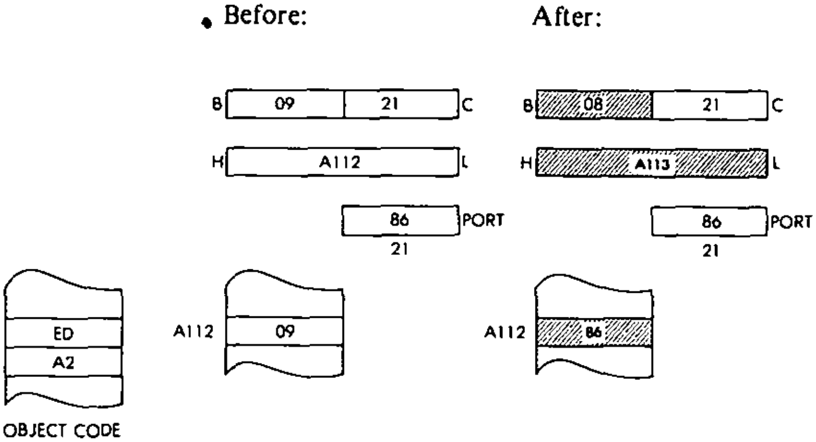
Addressing Mode: External.

Flags:

S	Z		H		P/V	N	C
?	X		?		?	1	

Z is set if B = 0 after execution,
Reset otherwise

Example: INI


Before:
After:
B
09
21
C
H
A112
L
86
PORT
21
B
08
21
C
H
A113
L
86
PORT
21
ED
A2
OBJECT CODE
A112
09
A112
86

INIR

Block input with increment.

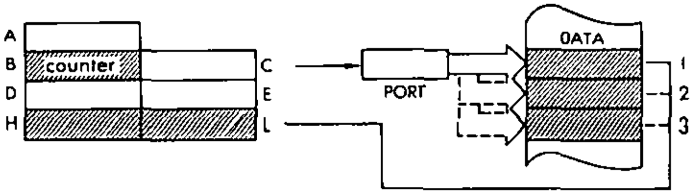
Function: (HL) ← (C); B ← B − 1; HL ← HL + 1; Repeat until B = 0

Format:

1	1	1	0	1	1	0	1	byte 1: ED
1	0	1	1	0	0	1	0	byte 2: B2

Description: The peripheral device addressed by the C register is read and the result is loaded into the memory location addressed by the HL register pair. The B register is decremented and the HL register pair is incremented. If B is not zero, the program counter is decremented by 2 and the instruction is re-executed.

Data Flow:



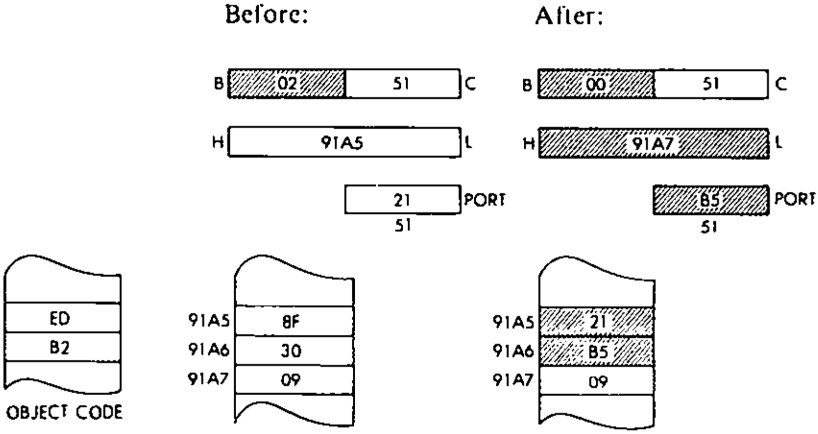
Timing: B = 0: 4 M cycles; 16 T states; 8 used @ 2 MHz.
B ≠ 0: 5 M cycles; 21 T states; 10.5 usec @ 2 MHz.

Addressing Mode: External.

Flags:

S	Z		H		P/V	N	C
?	1		?		?	1	

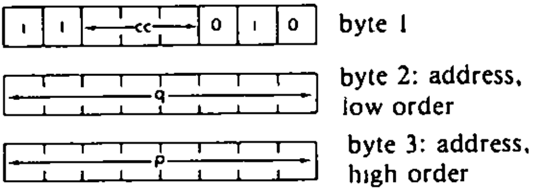
Example: INIR



JP cc, pq Jump on condition to location pq.

Function: if cc true: PC ← pq

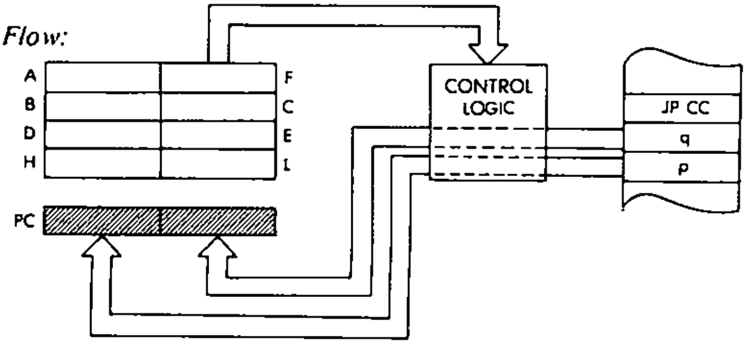
Format:



Description: If the specified condition is true, the two-byte address immediately following the opcode will be loaded into the program counter with the first byte following the opcode being loaded into the low order of the PC. If the condition is not met, the address is ignored. cc may be any one of:

NZ	– 000	no zero
Z	– 001	zero
NC	– 010	no carry
C	– 011	carry
PO	– 100	parity odd
PE	– 101	parity even
P	– 110	plus
M	– 111	minus

Data Flow:



Timing: 3 M cycles; 10 T states; 5 usec @ 2 MHz

Addressing Mode: Immediate.

Byte Codes:

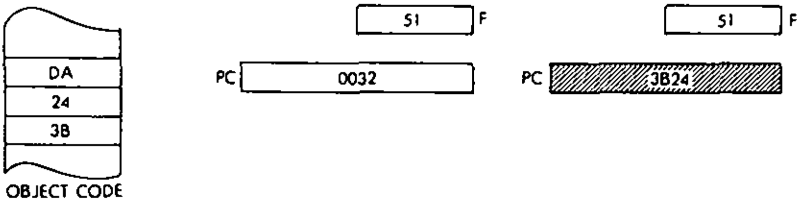
C C	NZ	Z	NC	C	PO	PE	P	M
	C2	CA	D2	DA	E2	EA	F2	FA

Flags:

S	Z		H		P/V	N	C	
								(no effect)

Example: JP C, 3B24

Before: After:

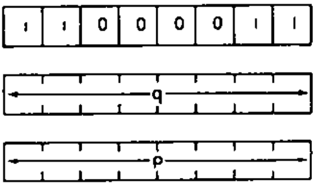


JP pq

Jump to location pq.

Function: PC ← pq

Format:



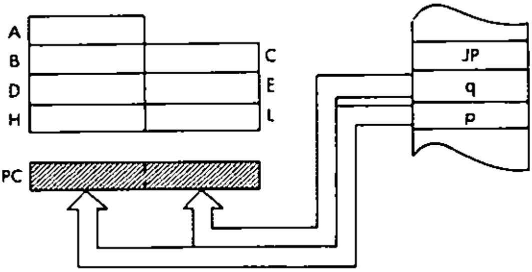
byte 1: C3

byte 2: address, low order

byte 3: address, high order

Description: The contents of the memory location immediately following the opcode are loaded into the low order half of the program counter and the contents of the second memory location immediately following the opcode are loaded into the high order of the program counter. The next instruction will be fetched from this new address.

Data Flow:



Timing: 3 M cycles; 10 T states; 5 usec @ 2 MHz

Addressing Mode: Immediate.

Flags:

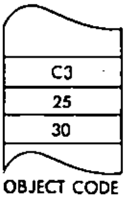
S	Z		H		P/V	N	C

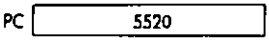
(No effect)

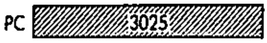
Example: JP 3025

Before:

After:







JP (HL)

Jump to HL.

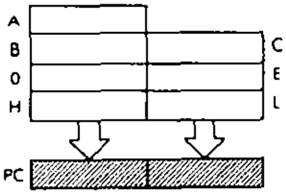
Function: PC ← HL

Format:

1	1	1	0	1	0	0	1	E9

Description: The contents of the HL register pair are loaded into the program counter. The next instruction is fetched from this new address.

Data Flow:



Timing: 1 M cycle; 4 T states; 2 usec @ 2 MHz

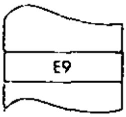
Addressing Mode: Implicit.

Flags:

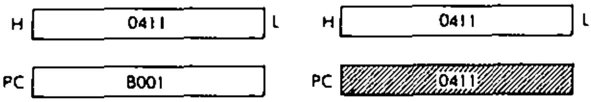
S	Z		H		P/V	N	C	
								(no effect).

Example: JP (HL)

Before: After:



OBJECT CODE



JP (IX) Jump to IX.

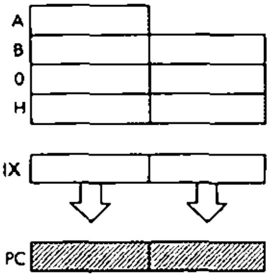
Function: PC ← IX

Format:

1	1	0	1	1	1	0	1	byte 1: DD
1	1	1	0	1	0	0	1	byte 2: E9

Description: The contents of the IX register are loaded into the program counter. The next instruction is fetched from this new address.

Data Flow:



Timing: 2 M cycles; 8 T states; 4 usec @ 2 MHz

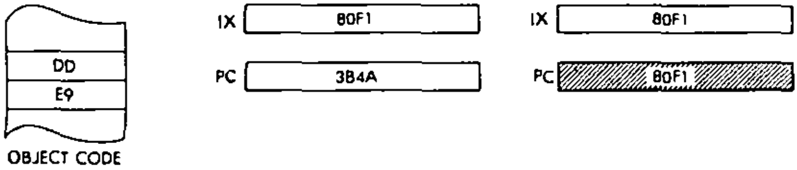
Addressing Mode: Implicit.

Flags:

S	Z		H		P/V	N	C	
								(no effect).

Example: JP (IX)

Before: After:



JP (IY)

Jump to IY.

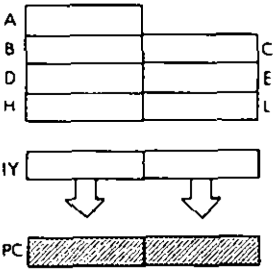
Function: PC ← IY

Format:

1	1	1	1	1	1	0	1	byte 1: FD
1	1	1	0	1	0	0	1	byte 2: E9

Description: The contents of the IY register are moved into the program counter. The next instruction will be fetched from this new address.

Data Flow:



Timing: 2 M cycles; 8 T states; 4 usec @ 2 MHz

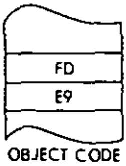
Addressing Mode: Implicit.

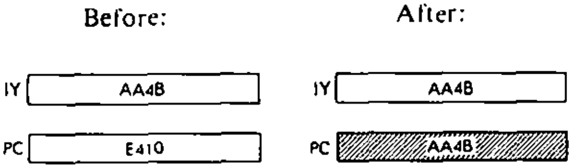
Flags:

S	Z		H		P/V	N	C

(no effect).

Example: JP (IY)

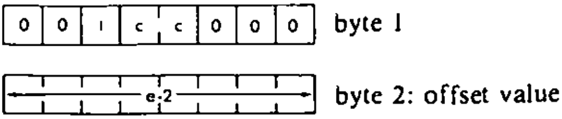




JR cc, e Jump e relative on condition.

Function: if cc true, PC ← PC + e

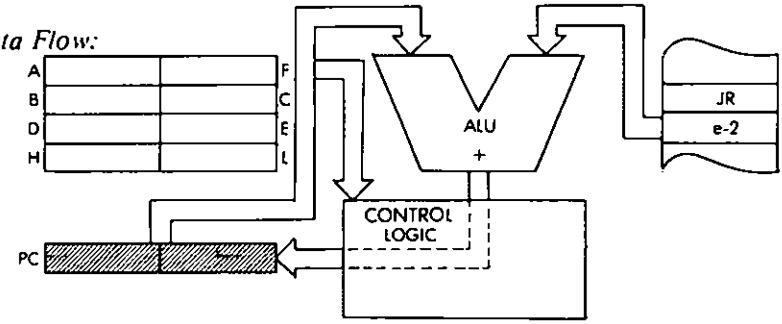
Format:



Description: If the specified condition is met, the given offset value is added to the program counter using two's complement arithmetic so as to enable both forward and backward jumps. The offset value is added to the value of PC + 2 (after the jump). As a result, the effective offset is -126 to +129 bytes. The assembler automatically subtracts 2 from the source offset value to generate the hex code. If the condition is not met, the offset value is ignored and instruction execution continues in sequence. cc may any one of:

NZ – 00	NC – 10
Z – 01	C – 11

Data Flow:



Timing:

	M cycles:	*T states:*	usec @ 2 MHz:
condition met:	3	12	6
condition not met:	2	7	3.5

Addressing Mode: Relative.

Byte Codes:

cc:	NZ	Z	NC	C
	20	28	30	38

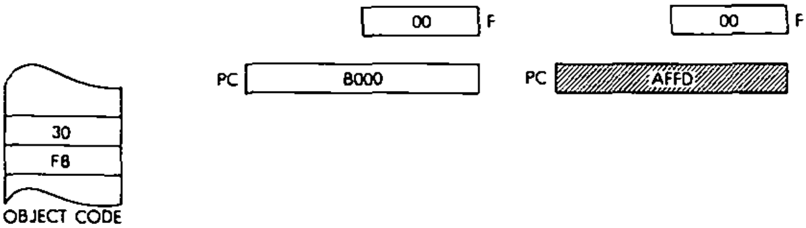
Flags:

S	Z		H		P/V	N	C

(no effect).

Example: JR NC, $ − 3 $ = current PC

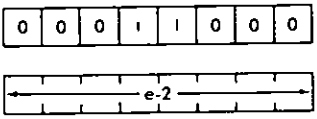
Before: Afier:



JR e Jump e relative.

Function: PC ← PC + e

Format:

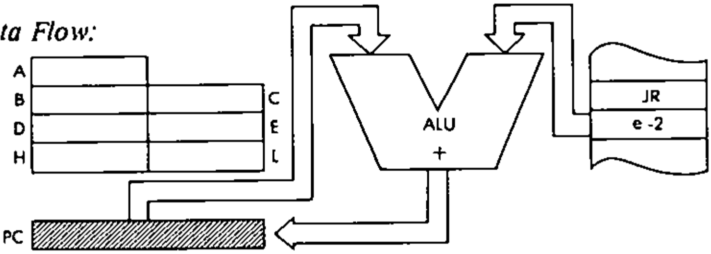


byte 1: 18

byte 2: offset value

Description: The given offset value is added to the program counter using two's complement arithmetic so as to enable both forward and backward jumps. The offset value is added to the value of PC + 2 (after the jump). As a result, the effective offset is -126 to + 129 bytes. The assembler automatically subtracts 2 from the source offset value to generate the hex code.

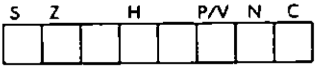
Data Flow:



Timing: 3 M cycles; 12 T states; 6 usec @ 2 MHz

Addressing Mode: Relative.

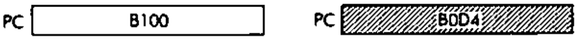
Flags:

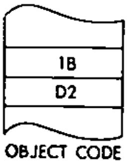


(no effect)

Example: JR D4

Before: After:





(This is a backwards jump.)

LD dd, (nn)

Load register pair dd from memory locations addressed by nn.

Function: $dd_{low} \leftarrow (nn);\ dd_{high} \leftarrow (nn+1)$

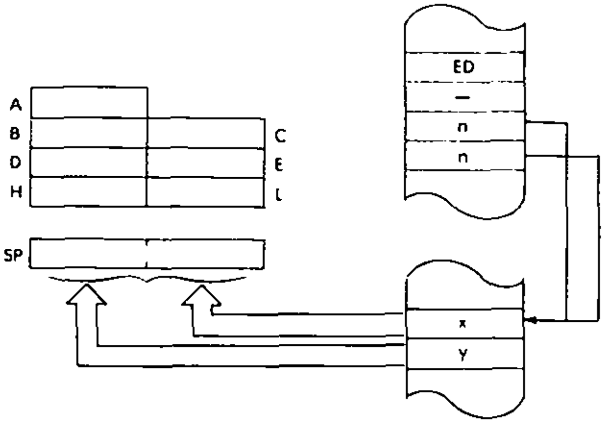
Format:

1	1	1	0	1	1	0	1	byte 1: ED
0	1	d	d	1	0	1	1	byte 2
← n →								byte 3: address, low order
← n →								byte 4: address, high order

Description: The contents of the memory location addressed by the memory locations immediately following the opcode are loaded into the low order of the specified register pair. The contents of the memory location immediately following the one previously loaded are then loaded into the high order of the register pair. The low order byte of the nn address immediately follows the opcode. dd may be any one of:

BC	00	HL	10
DE	01	SP	11

Data Flow:



Timing: 6 M cycles; 20 T states; 10 usec @ 2 MHz

Addressing Mode: Direct.

Byte Codes:

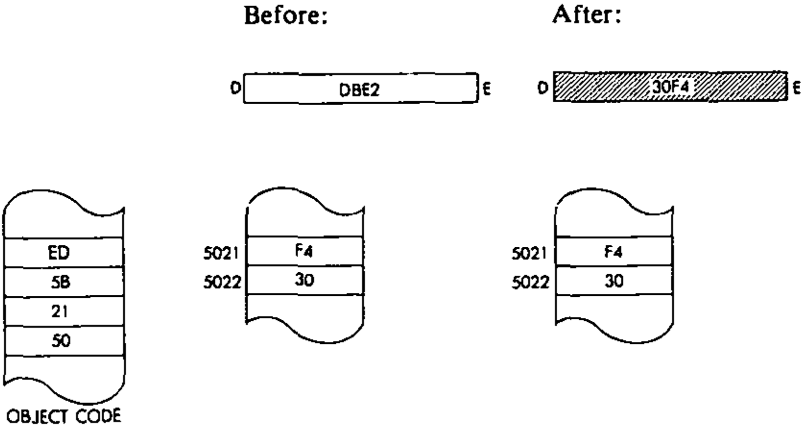
dd:	BC	DE	HL	SP
ED-	4B	5B	6B	7B

Flags:

S	Z		H		P/V	N	C

(no effect)

Example: LD DE, (5021)

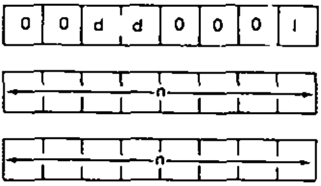


LD dd, nn

Load register pair dd with immediate data nn.

Function: dd ← nn

Format:



byte 1

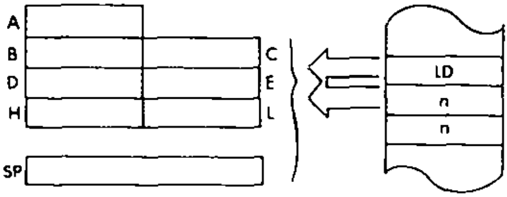
byte 2: immediate data, low order

byte 3: immediate data, high order

Description: The contents of the two memory locations immediately following the opcode are loaded into the specified register pair. The lower order byte of the data occurs immediately after the opcode. dd may be any one of:

BC	– 00	HL	– 10
DE	– 01	SP	– 11

Data Flow:



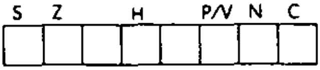
Timing: 3 M cycles; 10 T states; 5 usec @ 2 MHz

Addressing Mode: Immediate.

Byte Codes:

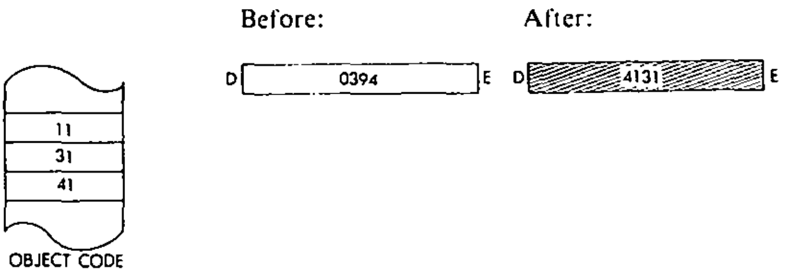
dd:	BC	DE	HL	SP
	01	11	21	31

Flags:



(no effect)

Example: LD DE, 4131

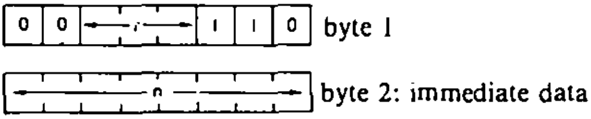


LD r, n

Load register r with immediate data n.

Function: r ← n

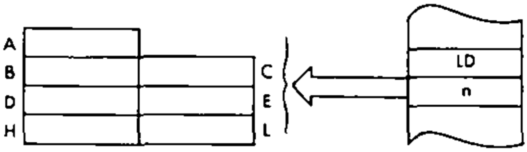
Format:



Description: The contents of the memory location immediately following the opcode location are loaded into the specified register. r may be any one of:

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

Data Flow:



Timing: 2 M cycles; 7 T states; 3.5 usec @ 2 MHz

Addressing Mode: Immediate.

Byte Codes:

r:	A	B	C	D	E	H	L
	3E	06	0E	16	1E	26	2E

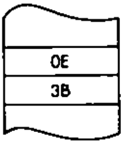
Flags:

S	Z		H		P/V	N	C

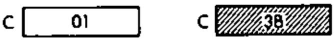
(no effect).

Example: LD C, 3B

Before: After:



OBJECT CODE

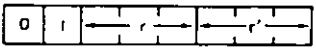


LD r, r'

Load register r from register r'.

Function: r ← r'

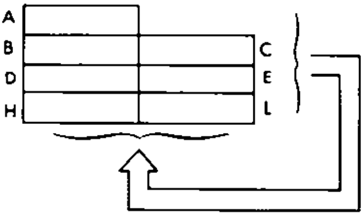
Format:



Description: The contents of the specified source register are loaded into the specified destination register. r and r' may be any one of:

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

Data Flow:



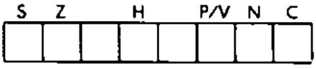
Timing: 1 M cycle; 4 T states; 2 usec @ 2 MHz

Addressing Mode: Implicit.

Byte Codes:

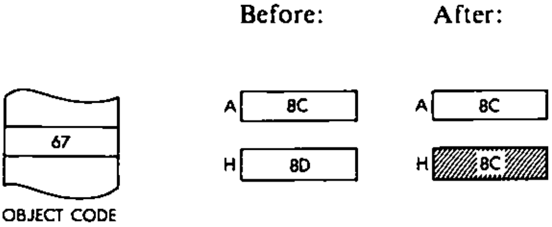
(dest.) \ (source)	A	B	C	D	E	H	L
A	7F	78	79	7A	78	7C	7D
B	47	40	41	42	43	44	45
C	4F	48	49	4A	4B	4C	4D
D	57	50	51	52	53	54	55
E	5F	58	59	5A	5B	5C	5D
H	67	60	61	62	63	64	65
L	6F	68	69	6A	68	6C	6D

Flags:



(no effect).

Example: LD H, A



LD (BC), A

Load indirectly addressed memory location (BC) from the accumulator.

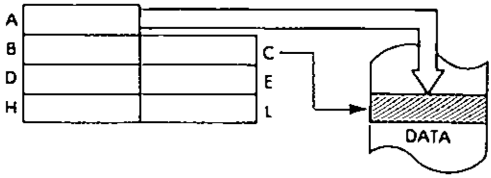
Function: (BC) ← A

Format:

0	0	0	0	0	0	1	0	02

Description: The contents of the accumulator are loaded into the memory location addressed by the contents of the BC register pair.

Data Flow:



Timing: 2 M cycles; 7 T states; 3.5 usec @ 2 MHz

Addressing Mode: Indirect.

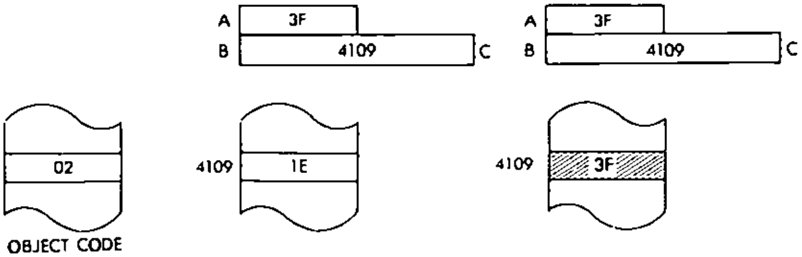
Flags:

S	Z		H		P/V	N	C

(no effect).

Example: LD (BC), A

Before: After:



LD (DE), A

Load indirectly addressed memory location (DE) from the accumulator.

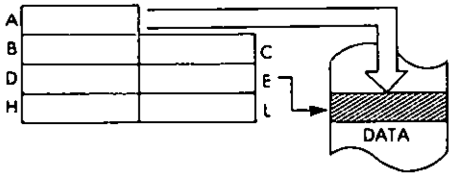
Function: (DE) ← A

Format:

0	0	0	1	0	0	1	0	12

Description: The contents of the accumulator are loaded into the memory location addressed by the contents of the DE register pair.

Data Flow:



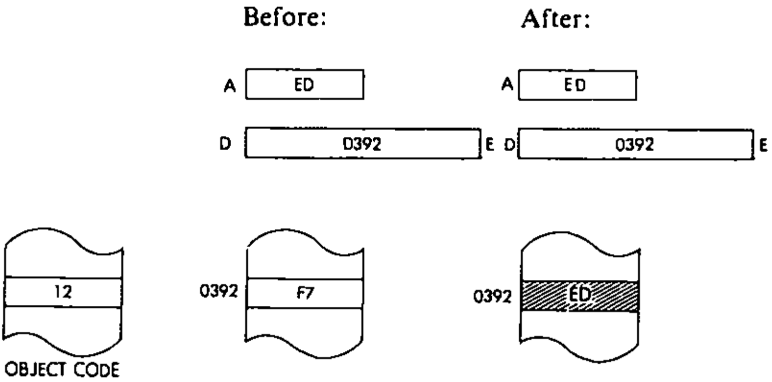
Timing: 2 M cycles; 7 T states; 3.5 usec @ 2 MHz

Addressing Mode: Indirect.

Flags:

S	Z		H		P/V	N	C	
								(no effect)

Example: LD (DE), A



LD (HL), n Load immediate data n into the indirectly addressed memory location (HL).

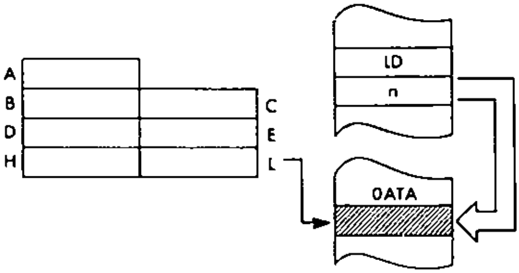
Function: (HL) ← n

Format:

0	0	1	1	0	1	1	0	byte 1: 36
←			n				→	byte 2: immediate data

Description: The contents of the memory location immediately following the opcode are loaded into the memory location indirectly addressed by the HL data pointer

Data Flow:



Timing: 3 M cycles; 10 T states; 5 usec @ 2 MHz

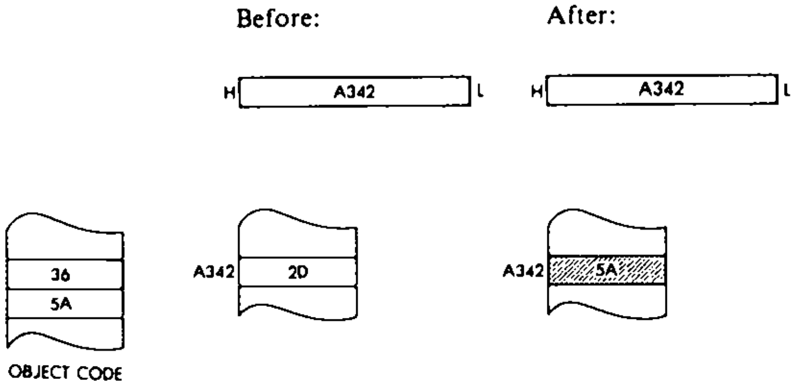
Addressing Mode: Immediate/indirect.

Flags:

S	Z		H		P/V	N	C

(no effect).

Example: LD (HL), 5A



LD (HL), r

Load indirectly addressed memory location (HL) from register r.

Function: (HL) ← r

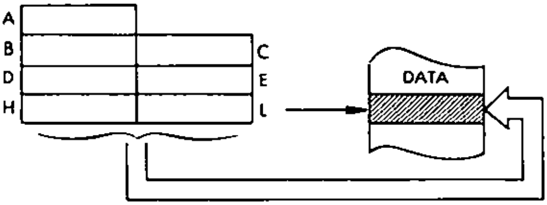
Format:

0	1	1	1	0	← r →

Description: The contents of the specified register are loaded into the memory location addressed by the HL register pair. r may be any one of:

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

Data Flow:



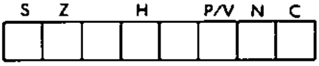
Timing: 2 M cycles; 7 T states; 3.5 usec @ 2 MHz

Addressing Mode: Indirect.

Byte Codes:

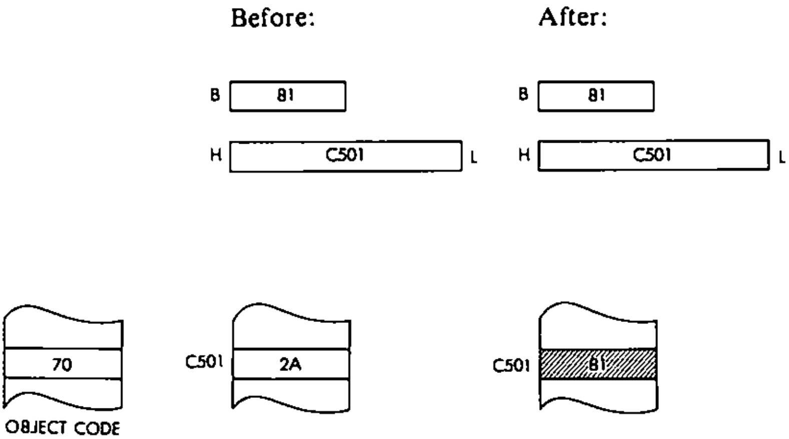
r:	A	B	C	D	E	H	L
	77	7D	71	72	73	74	75

Flags:



(no effect).

Example: LD (HL), B



LD r, (IX + d) Load register r indirect from indexed memory location (IX + d)

Function: r ← (IX + d)

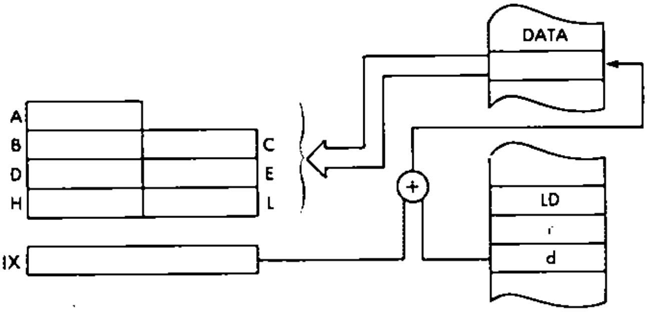
Format:

1	1	0	1	1	1	0	1	byte 1: DD
0	1	← r →			1	1	0	byte 2
← d →								byte 3: offset value

Description: The contents of the memory location addressed by the IX index register plus the given offset value, are loaded into the specified register. r may be any one of:

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

Data Flow:



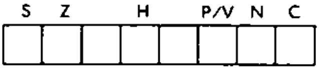
Timing: 5 M cycles; 19 T states; 9.5 usec @ 2 MHz

Addressing Mode: Indexed.

Byte Codes:

r:	A	B	C	D	E	H	L	
DD-	7E	46	4E	56	5E	66	6E	-d

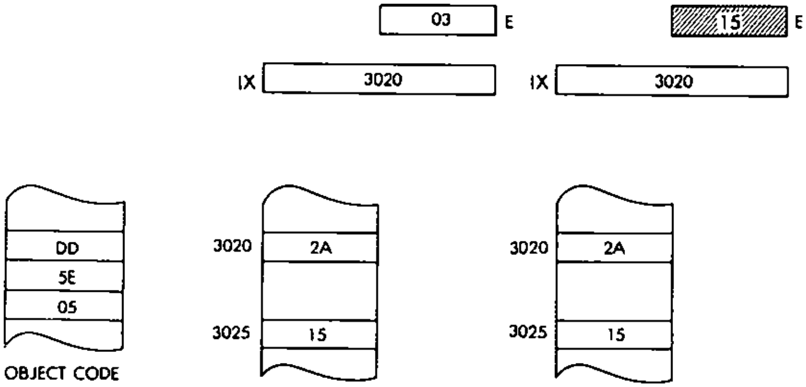
Flags:



(no effect).

Example: LD E, (IX + 5)

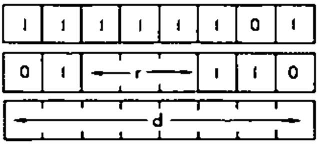
Before: After:



LD r, (IY + d) Load register r indirect from indexed memory location (IY + d)

Function: r ← (IY + d)

Format:



byte 1: FD

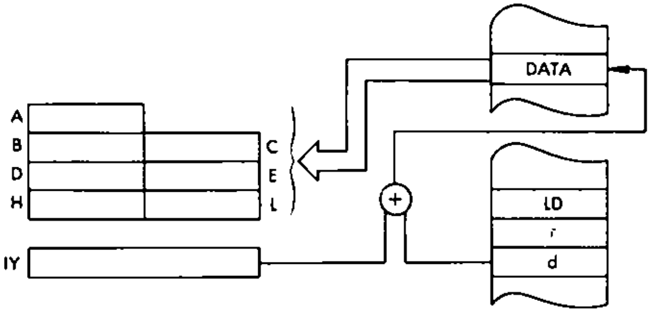
byte 2

byte 3: offset value

Description: The contents of the memory location addressed by the IY index register plus the given offset value, are loaded into the specified register. r may be any one of:

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

Data Flow:



Timing: 5 M cycles, 19 T states; 9.5 usec @ 2 MHz

Addressing Mode: Indexed.

Byte Codes:

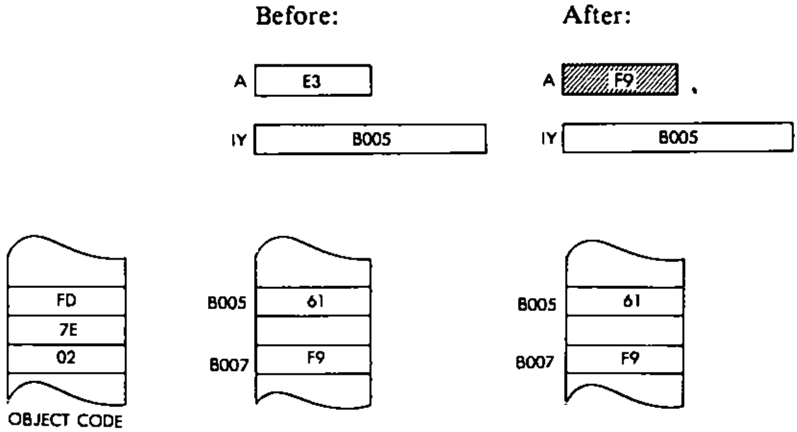
r:	A	B	C	D	E	H	L	
FD -	7E	46	4E	56	56	66	6E	- d

Flags:

S	Z		H		P/V	N	C

(no effect).

Example: LD A, (IY + 2)



LD (IX + d), n Load indexed addressed memory location (IX + d) with immediate data n.

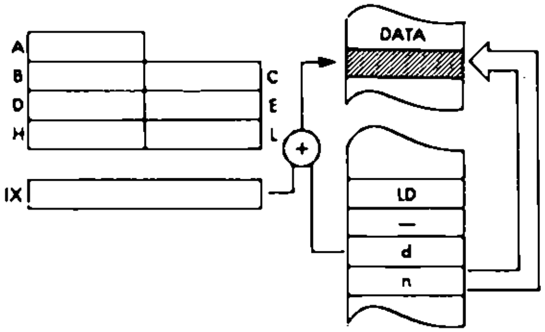
Function: (IX + d) ← n

Format:

1	1	0	1	1	1	0	1	byte 1: DD
0	0	1	1	0	1	1	0	byte 2: 36
← d →								byte 3: offset value
← n →								byte 4: immediate data

Description: The contents of the memory location immediately following the offset are transferred into the memory location addressed by the contents of the index register plus the given offset value.

Data Flow:



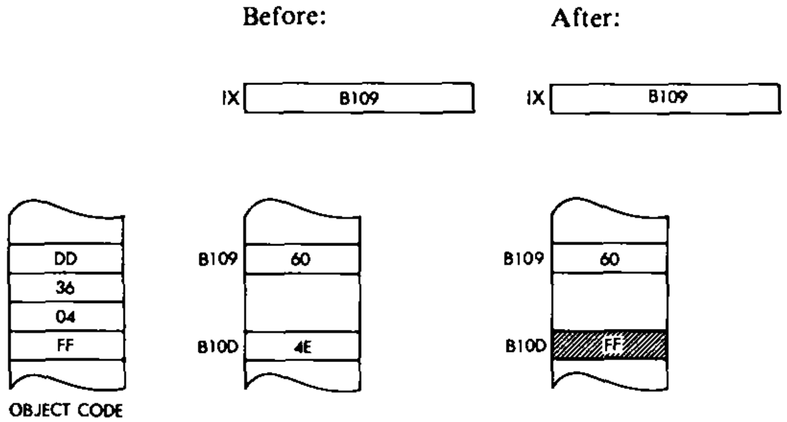
Timing: 5 M cycles; 19 T states; 9.5 usec @ 2 MHz

Addressing Mode: Indexed/immediate.

Flags:

S	Z		H		P/V	N	C	
								(no effect).

Example: LD (IX + 4), FF



LD (IY + d), n Load indexed addressed memory location (IY + d) with immediate data n.

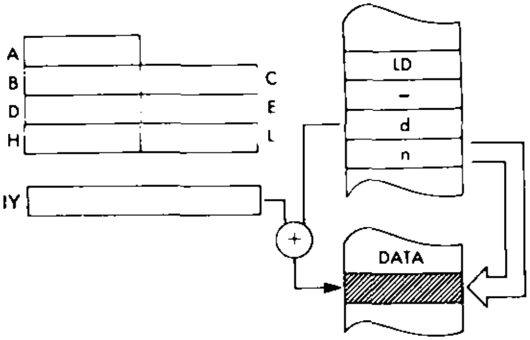
Function: (IY + d) ← n

Format:

1	1	1	1	1	1	0	1	byte 1: FD
0	0	1	1	0	1	1	D	byte 2: 36
←			d				→	byte 3: offset value
←			n				→	byte 4: immediate data

Description: The contents of the memory location immediately following the offset are transferred into the memory location addressed by the contents of the index register plus the given offset value.

Data Flow:



Timing: 5 M cycles; 19 T states; 9.5 usec @ 2 MHz

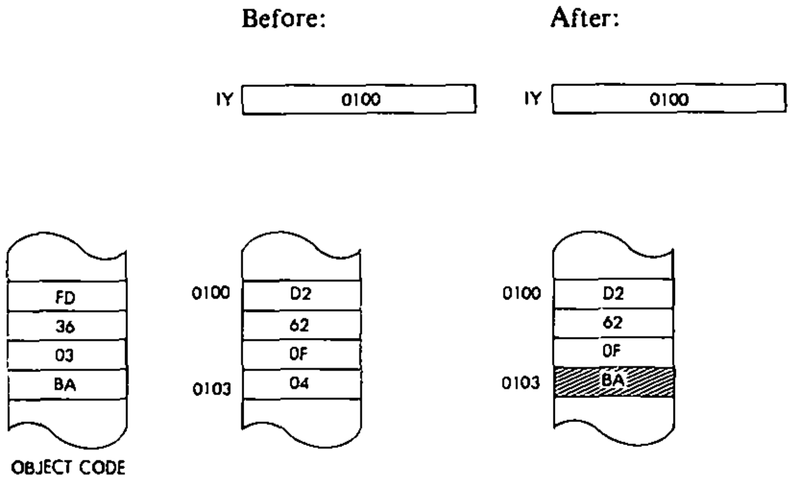
Addressing Mode: Indexed/immediate.

Flags:

S	Z		H		P/V	N	C

(no effect).

Example: LD (IY + 3), BA



LD (IX + d),r Load indexed addressed memory location (IX + d) from register r.

Function: (IX + d) ← r

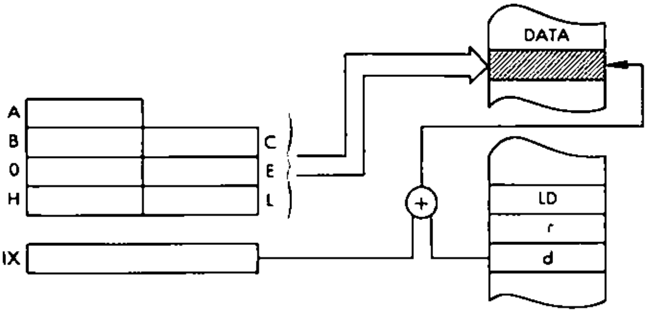
Format:

1	1	0	1	1	1	D	1	byte 1: DD
0	1	1	1	0	← r →			byte 2
← d →								byte 3: offset value

Description: The contents of specified register are loaded into the memory location addressed by the contents of the index register plus the given offset value. r may be any one of:

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

Data Flow:



Timing: 5 M cycles; 19 T states; 9.5 usec @ 2 MHz

Addressing Mode: Indexed.

Byte Codes:

r:	A	B	C	D	E	H	L	
DD-	77	70	71	72	73	74	75	- d

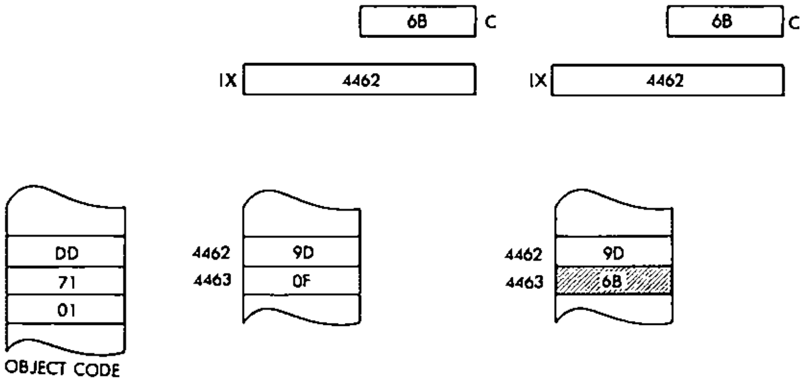
Flags:

S	Z		H		P/V	N	C

(no effect).

Example: LD (IX + I), C

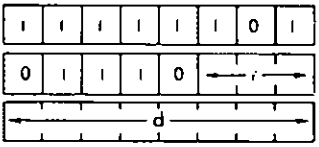
Before: After:



LD (IY + d), r Load indexed addressed memory location (IY + d) from register r.

Function: (IY + d) ← r

Format:



byte 1: FD

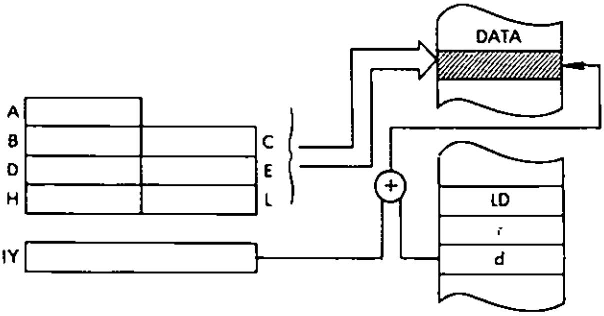
byte 2

byte 3: offset value

Description: The contents of the specified register are loaded into the memory location addressed by the contents of the index register plus the given offset value. r may be any one of:

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

Data Flow:



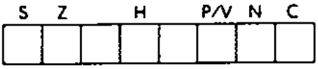
Timing: 5 M cycles; 19 T states; 9.5 usec @ 2 MHz

Addressing Mode: Indexed.

Byte Codes:

r:	A	B	C	D	E	H	L	
FD-	77	70	71	72	73	74	75	-d

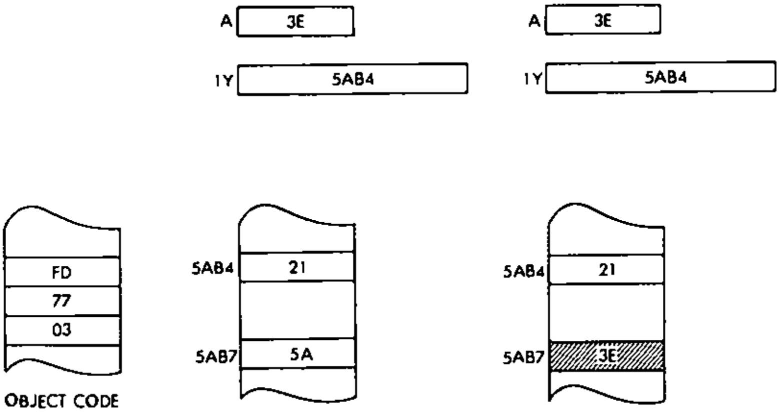
Flags:



(no effect).

Example: LD (IY + 3), A

Before: After:



LD A, (nn) Load accumulator from the memory location (nn).

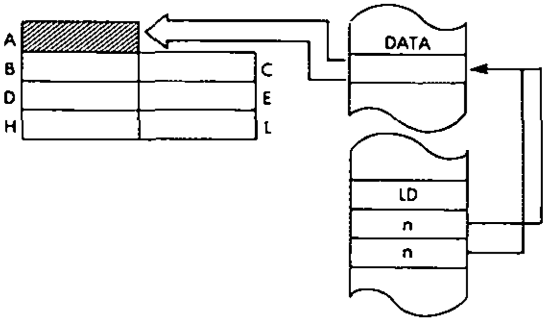
Function: A ← (nn)

Format:

0	0	1	1	1	0	1	0	byte 1: 3A
← n →								byte 2: address, low order byte
← n →								byte 3: address, high order byte

Description: The contents of the memory location addressed by the contents of the 2 memory locations immediately following the opcode are loaded into the accumulator. The low byte of the address occurs immediately after the opcode.

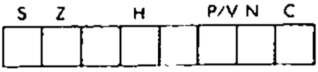
Data Flow:



Timing: 4 M cycles; 13 T states; 6.5 usec @ 2 MHz

Addressing Mode: Direct.

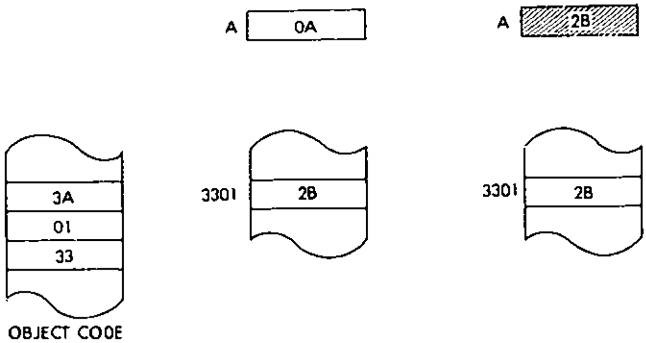
Flags:



(no effect).

Example: LD A, (3301)

Before: After:

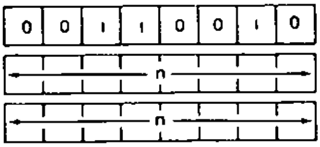


LD (nn),A

Load directly addressed memory location (nn) from accumulator.

Function: (nn) ← A

Format:



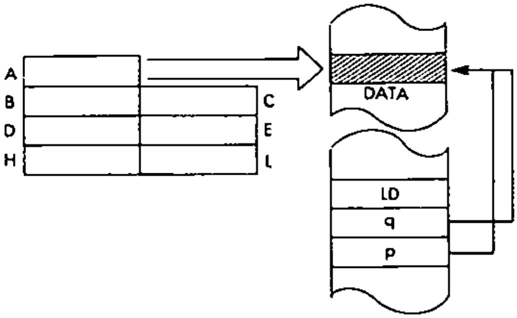
byte 1: 32

byte 2: address, low order

byte 3: address, high order

Description: The contents of the accumulator are loaded into the memory location addressed by the contents of the memory locations immediately following the opcode. The low byte of the address immediately follows the opcode.

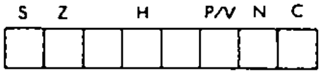
Data Flow:



Timing: 4 M cycles; 13 T states; 6.5 usec @ 2 MHz

Addressing Mode: Direct.

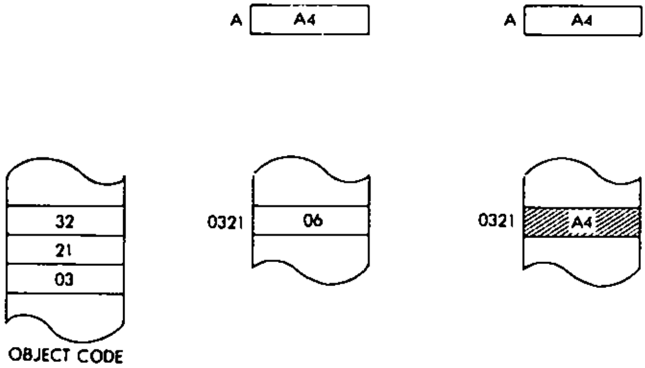
Flags:



(no effect)

Example: LD (0321), A

Before: After:



LD (nn), dd Load memory locations addressed by nn from register pair rr.

Function: $(nn) \leftarrow dd_{low}; (nn + 1) \leftarrow dd_{high}$

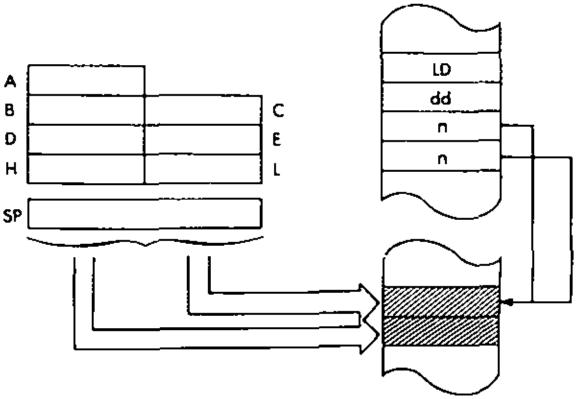
Format:

1	1	1	0	1	1	0	1	byte 1: ED
0	1	d	d	0	0	1	1	byte 2
n								byte 3: address, low order
n								byte 4: address, high order

Descriptions: The contents of the low order of the specified register pair are loaded into the memory location addressed by the memory locations immediately following the opcode. The contents of the high order of the register pair are loaded into the memory location immediately following the one loaded from the low order. The low order of the nn address occurs immediately after the opcode.dd may be anyone of:

BC – 00	HL – 10
DE – 01	SP – 11

Data Flow:



Timing: 6 M cycles; 20 T states; 10 usec @ 2 MHz

Addressing Mode: Direct.

Byte Codes:

dd:	BC	DE	HL	SP
ED-	43	53	63	73

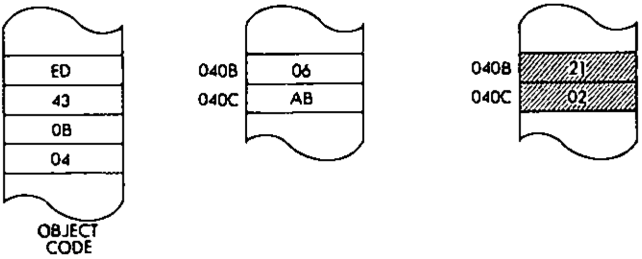
Flags:

S	Z		H		P/V	N	C

(no effect).

Example: LD (040B), BC

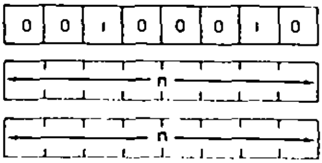
Before:		After:	
B	0221 C	B	0221 C



LD (nn), HL Load the memory locations addressed by nn from HL.

Function: (nn) ← L; (nn + 1) ← H

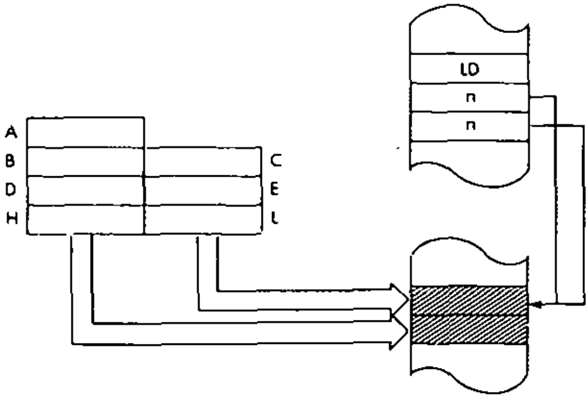
Format:



byte 1: 22
byte 2: address, low order
byte 3: address, high order

Description: The contents of the L register are loaded into the memory location addressed by the memory locations immediately following the opcode. The contents of the H register are loaded into the memory location immediately following the location loaded from the L register. The low order of the nn address occurs immediately after the opcode.

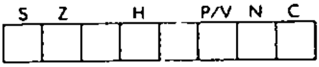
Data Flow:



Timing: 5 M cycles; 16 T states; 8 usec @ 2 MHz

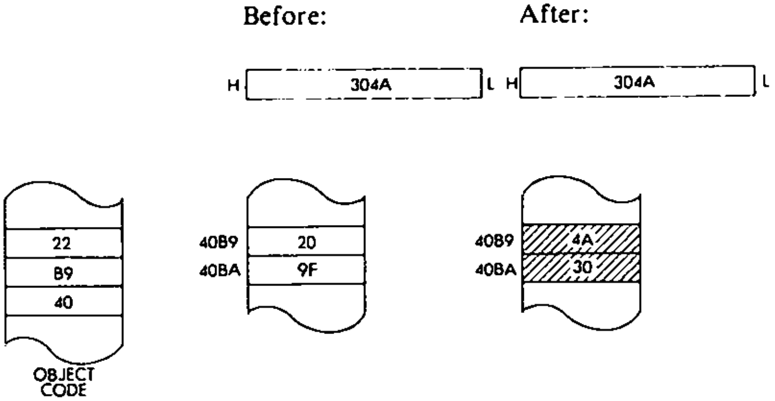
Addressing Mode: Direct.

Flags:



(no effect).

Example: LD (40B9), HL



LD (nn), IX

Load memory locations addressed by nn from IX.

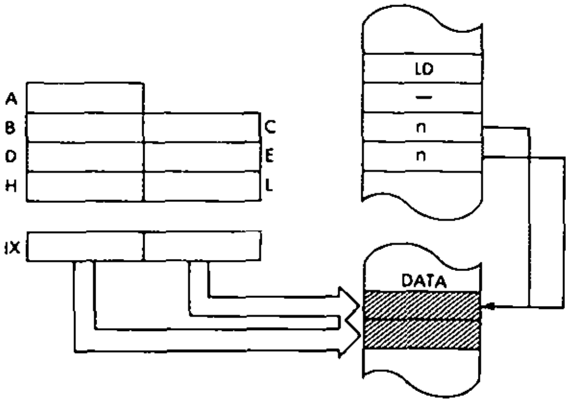
Function: (nn) ← IX_{low}; (nn + 1) ← IX_{high}

Format:

1	1	0	1	1	1	0	1	byte 1: DD
0	0	1	0	0	0	1	0	byte 2: 22
n								byte 3: address, low order
n								byte 4: address, high order

Description: The contents of the low order of the IX register are loaded into the memory location addressed by the contents of the memory location immediately following the opcode. The contents of the high order of the IX register are loaded into the memory location immediately following the one loaded from the low order. The low order of the nn address occurs immediately after the op code.

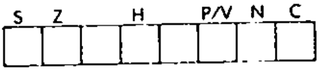
Data Flow:



Timing: 6 M cycles; 20 T states; 10 usec @ 2 MHz

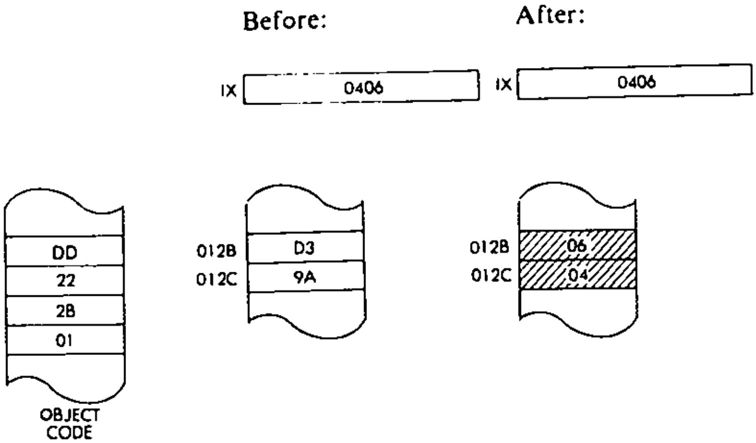
Addressing Mode: Direct.

Flags:



(no effect).

Example: LD (012B), IX



LD (nn), IY Load memory locations addressed by nn from IY.

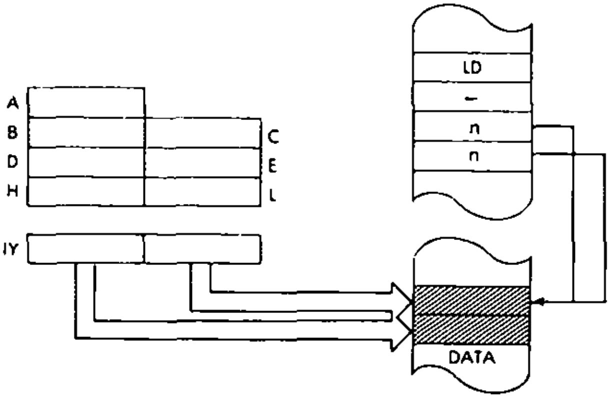
Function: $(nn) \leftarrow IY_{low}; (nn + 1) \leftarrow IY_{high}$

Format:

1	1	1	1	1	1	0	1	byte 1: FD
0	0	1	0	0	0	1	0	byte 2: 22
← n →								byte 3: address, low order
← n →								byte 4: address, high order

Description: The contents of the low order of the IY register are loaded into the memory location addressed by the contents of the memory locations immediately following the opcode. The contents of the high order of the IY register are loaded into the memory location immediately following the one loaded from the low order. The low order of the nn address occurs immediately after the opcode.

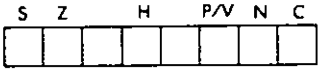
Data Flow:



Timing: 6 M cycles; 20 T states; 10 usec @ 2 MHz

Addressing Mode: Direct.

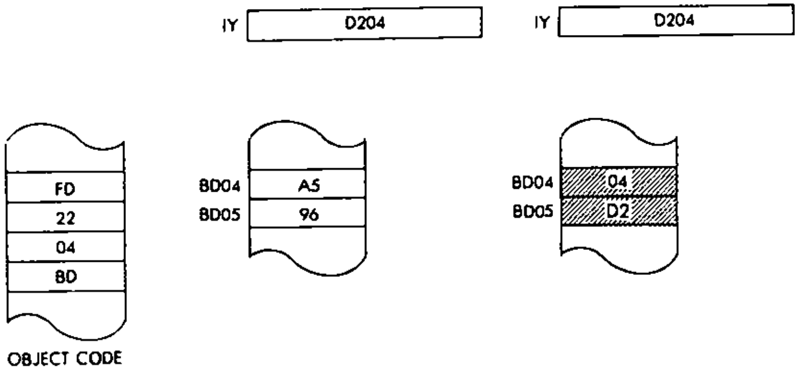
Flags:



(no effect)

Example: LD (BD04), IY

Before: After:



LD A, (BC)

Load accumulator from the memory location indirectly addressed by the BC register pair.

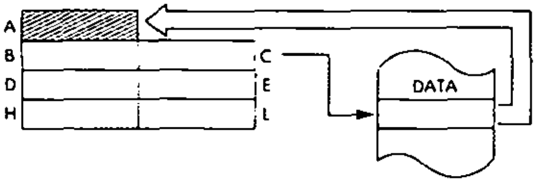
Function: A ← (BC)

Format:

0	0	0	0	1	0	1	0	0A

Description: The contents of the memory location addressed by the contents of the BC register pair are loaded into the accumulator.

Data Flow:



Timing: 2 M cycles; 7 T states; 3.5 usec @ 2 MHz

Addressing Mode: Indirect.

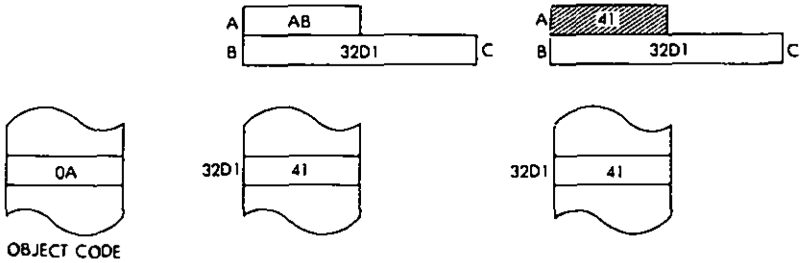
Flags:

S	Z		H		P/V	N	C

(no effect).

Example: LD A, (BC)

Before: After:



LD A, (DE) Load the accumulator from the memory location indirectly addressed by the DE register pair.

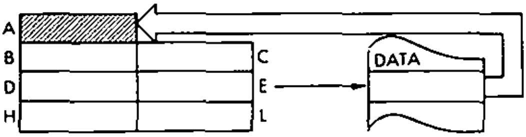
Function: A ← (DE)

Format:

0	0	0	1	1	0	1	0	1A

Description: The contents of the memory location addressed by the contents of the DE register pair are loaded into the accumulator.

Data Flow:



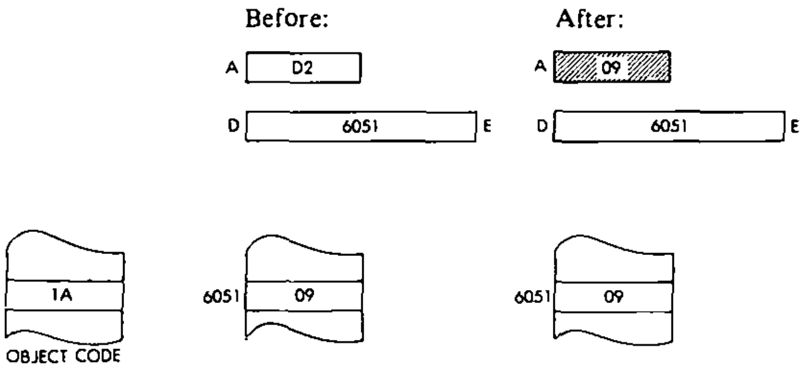
Timing: 2 M cycles; 7 T states; 3.5 usec @ 2 MHz

Addressing Mode: Indirect.

Flags:

S	Z		H		P/V	N	C	
								(No effect).

Example: LD A, (DE)



LD A, I

Load accumulator from interrupt vector register I.

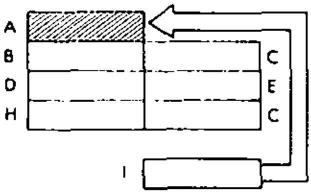
Function: A ← I

Format:

1	1	1	0	1	1	0	1	byte 1: ED
0	1	0	1	0	1	1	1	byte 2: 57

Description: The contents of the interrupt vector register are loaded into the accumulator.

Data Flow:



Timing: 2 M cycles; 9 T states; 4.5 usec @ 2 MHz

Addressing Mode: Implicit.

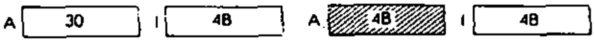
Flags:

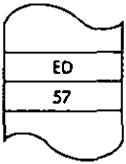
S	Z		H		P/V	N	C
●	●		○		×	○	

Set to the contents of IFF2

Example: LD A, I

Before: After:





OBJECT CODE

LD I, A

Load Interrupt Vector register I from the accumulator.

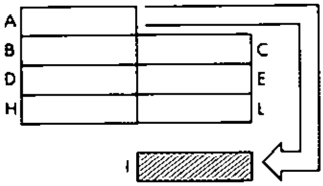
Function: I ← A

Format:

1	1	1	0	1	1	0	1	byte 1: ED
0	1	0	0	0	1	1	1	byte 2: 47

Description: The contents of the accumulator are loaded into the Interrupt Vector register.

Data Flow:



Timing: 2 M cycles; 9 T states; 4.5 usec @ 2 MHz

Addressing Mode: Implicit.

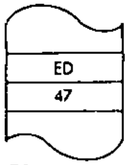
Flags:

S	Z		H		P/V	N	C	
								(no effect)

Example: LD I, A

Before: After:

A	06	I	D2	A	06	I	06



OBJECT CODE

LD A, R

Load accumulator from Memory Refresh register R.

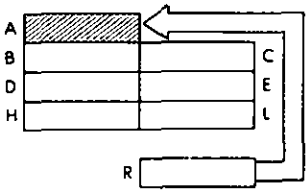
Function: A ← R

Format:

1	1	1	0	1	1	0	1	byte 1: ED
0	1	0	1	1	1	1	1	byte 2: 5F

Description: The contents of the Memory Refresh register are loaded into the accumulator.

Data Flow:



Timing: 2 M cycles; 9 T states; 4.5 usec @ 2 MHz

Addressing Mode: Implicit.

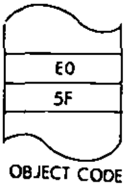
Flags:

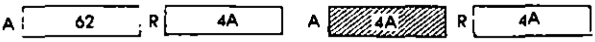
S	Z		H		P/V	N	C
●	●		○		×	○	

set to contents of IFF2

Example: LD A, R

Before: After:





LD HL, (nn) Load HL register from memory locations addressed by nn.

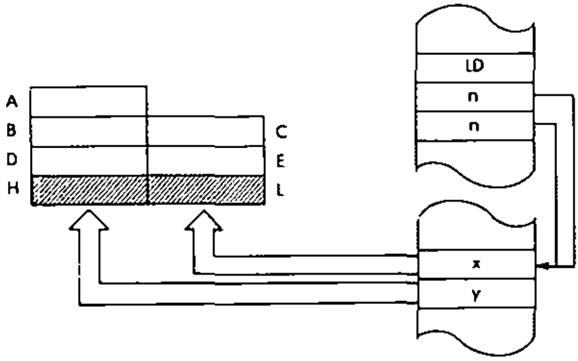
Function: L ← (nn); H ← (nn + 1)

Format:

0 0 1 0 1 0 1 0	byte 1: 2A
←— n —→	byte 2: address, low order
←— n —→	byte 3: address, high order

Description: The contents of the memory location addressed by the memory locations immediately after the opcode are loaded into the L register. The contents of the memory location after the one loaded into the L register are loaded into the H register. The low byte of the nn address occurs immediately after the opcode.

Data Flow:



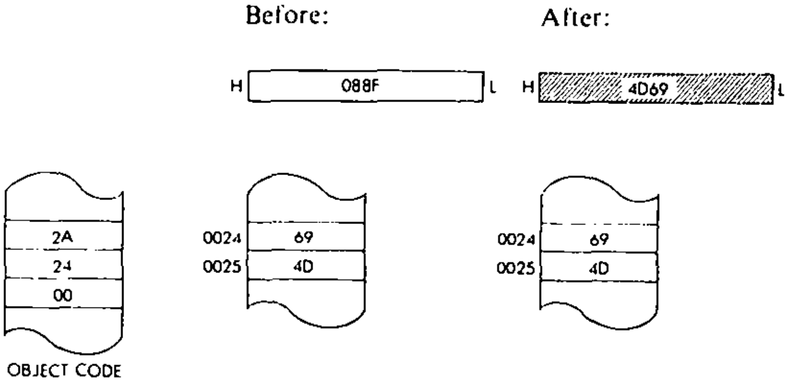
Timing: 5 M cycles, 16 T states; 8 usec @ 2 MHz

Addressing Mode: Direct.

Flags:

S Z H P/V N C
□□□□□□□□ (no effect)

Example: LD HL, (0024)



LD IX, nn Load IX register with immediate data nn.

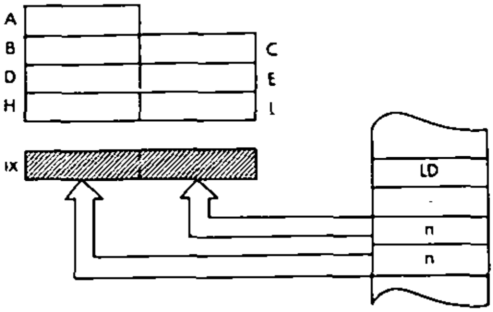
Function: IX ← nn

Format:

1	1	0	1	1	1	0	1	byte 1: DD
0	0	1	0	0	0	0	1	byte 2: 21
←			n				→	byte 3: immediate data, low order
←			n				→	byte 4: immediate data, high order

Description: The contents of the memory locations immediately following the opcode are loaded into the IX register. The low order byte occurs immediately after the opcode.

Data Flow:



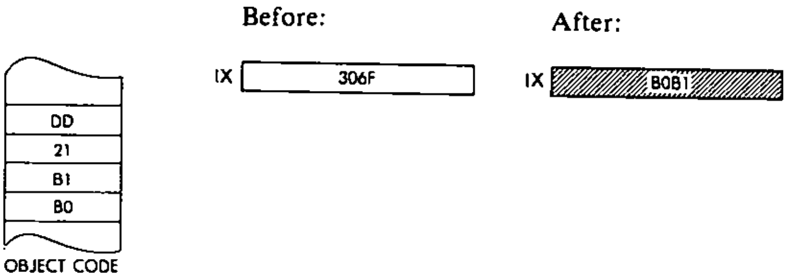
Timing: 4 M cycles; 14 T states; 7 usec @ 2 MHz

Addressing Mode: Immediate.

Flags:

S	Z		H		P/V	N	C	
								(no effect)

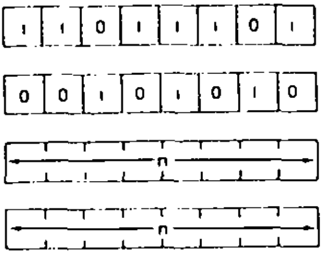
Example: LD IX, B0B 1



LD IX, (nn) Load IX register from memory locations addressed by nn.

Function: $IX_{low} \leftarrow (nn); IX_{high} \leftarrow (nn + 1)$

Format:



byte 1: DD

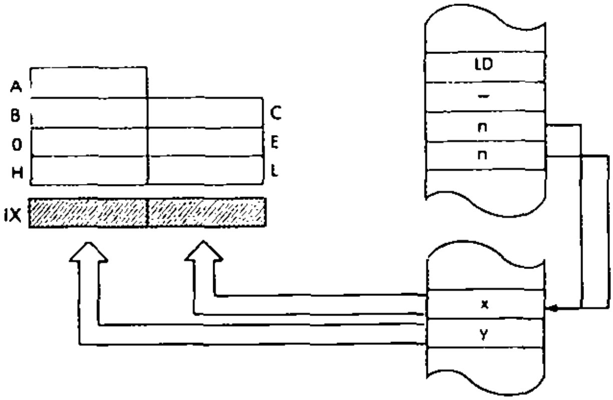
byte 2: 2A

byte 3: address, low order

byte 4: address, high order

Descriptions: The contents of the memory location addressed by the memory locations immediately following the opcode are loaded into the low order of the IX register. The contents of the memory location immediately following the one loaded into the low order are loaded into the high order of the IX register. The low order of the nn address immediately follows the opcode.

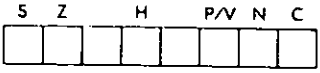
Data Flow:



Timing: 6 M cycles; 20 T states; 10 usec @ 2 MHz

Addressing Mode: Direct.

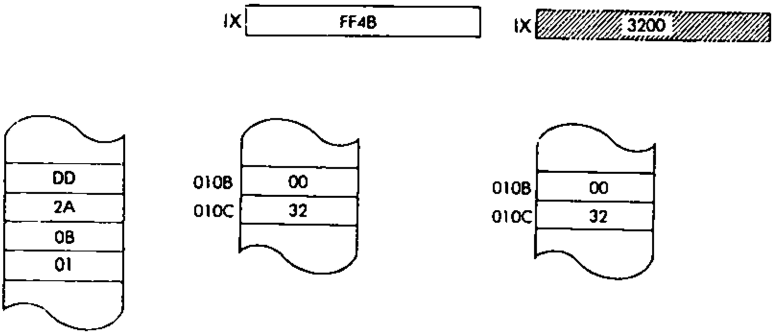
Flags:



(no effect).

Example: LD IX, (010B)

Before: After:

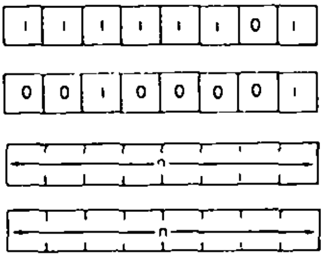


OBJECT CODE

LD IY, nn Load IY register with immediate data nn.

Function: IY ← nn

Format:



byte 1: FD

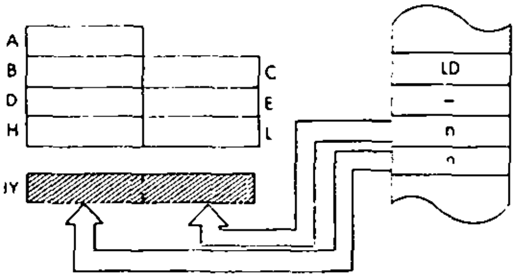
byte 2: 21

byte 3: immediate data, low order

byte 4: immediate data, high order

Description: The contents of the memory locations immediately following the opcode are loaded into the IY register. The low order byte occurs immediately after the opcode.

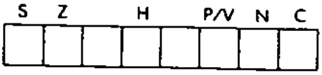
Data Flow:



Timing: 4 M cycles; 14 T states; 7 usec @ 2 MHz

Addressing Mode: Immediate.

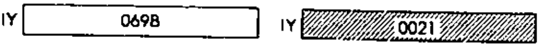
Flags:

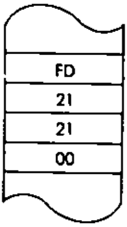


(no effect)

Example: LD IY, 21

Before: After:



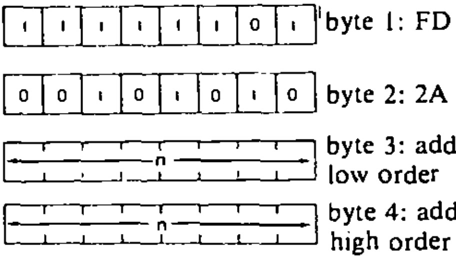


OBJECT CODE

LD IY, (nn) Load register IY from memory locations addressed by nn.

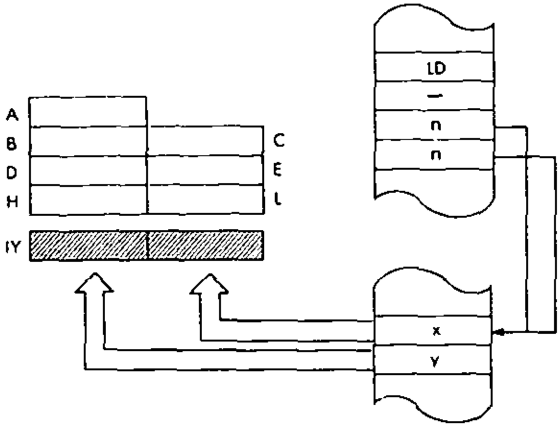
Function: $IY_{low} \leftarrow (nn); IY_{high} \leftarrow (nn + 1)$

Format:



Description: The contents of the memory location addressed by the memory locations immediately following the opcode are loaded into the low order of the IY register. The contents of the memory location immediately following the one loaded into the low order are loaded into the high order of the IY register. The low order of the nn address immediately follows the opcode.

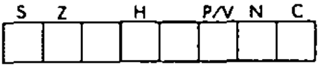
Data Flow:



Timing: 6 M cycles; 20 T states; 10 usec @ 2 MHz

Addressing Mode: Direct.

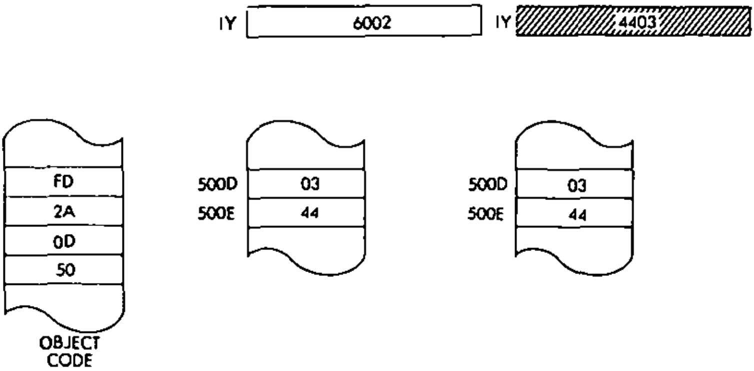
Flags:



(no effect).

Example: LD IY, (500D)

Before: After:



LD R,A

Load Memory Refresh register R from the accumulator.

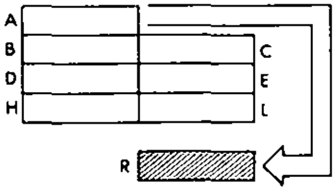
Function: R ← A

Format:

1	1	1	0	1	1	0	1	byte 1: ED
0	1	0	0	1	1	1	1	byte 2: 4F

Description: The contents of the accumulator are loaded into the Memory Refresh register.

Data Flow:



Timing: 2 M cycles; 9 T states; 4.5 usec @ 2 MHz

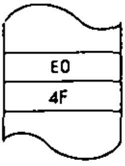
Addressing Mode: Implicit.

Flags:

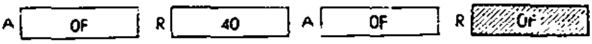
S	Z		H		P/V	N	C	
								(no effect)

Example: LD R, A

Before: After:



OBJECT CODE



LD SP, HL

Load stack pointer from HL.

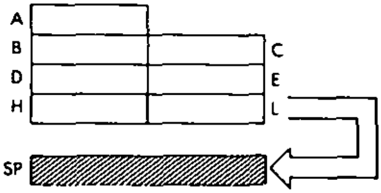
Function: SP ← HL

Format:

1	1	1	1	1	0	0	1	F9

Description: The contents of the HL register pair are loaded into the stack pointer.

Data Flow:



Timing: 1 M cycles; 6 T states; 3 usec @ 2 MHz

Addressing Mode: Implicit.

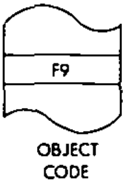
Flags:

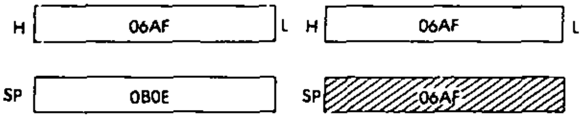
S	Z		H		P/V	N	C

(no effect)

Example: LD SP, HL

Before: After:





LD SP, IX

Load stack pointer from IX register.

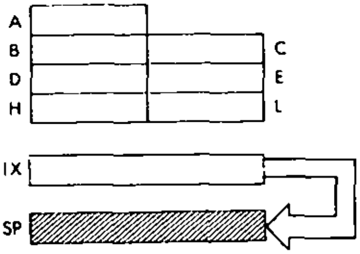
Function: SP ← IX

Format:

1	1	0	1	1	1	0	1	byte 1: DD
1	1	1	1	1	0	0	1	byte 2: F9

Description: The contents of the IX register are loaded into the stack pointer.

Data Flow:



Timing: 2 M cycles; 10 T states; 5 usec @ 2 MHz

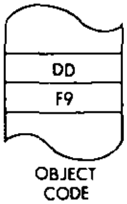
Addressing Mode: Implicit.

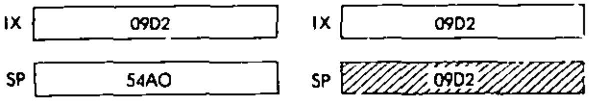
Flags:

S	Z		H		P/V	N	C	
								(no effect)

Example: LD SP, IX

Before: After:





LD SP, IY

Load stack pointer from IY register.

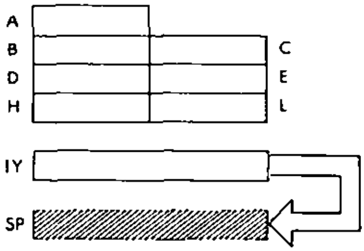
Function: SP ← IY

Format:

1	1	1	1	1	1	0	1	byte 1: FD
1	1	1	1	1	0	0	1	byte 2: F9

Description: The contents of the IY register are loaded into the stack pointer.

Data Flow:



Timing: 2 M cycles; 10 T states; 5 usec @ 2 MHz

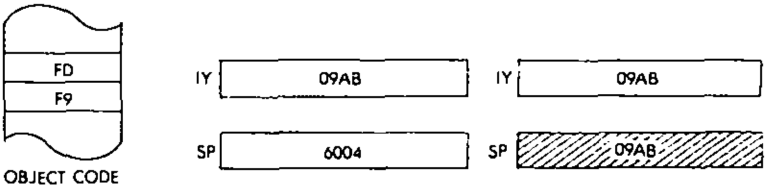
Addressing Mode: Implicit.

Flags:

S	Z		H		P/V	N	C	
								(no effect)

Example: LD SP, IY

Before: After:



LDD

Block load with decrement.

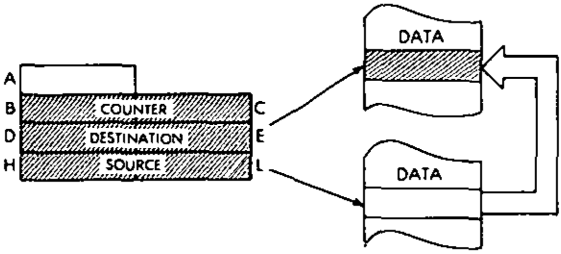
Function: (DE) ← (HL); DE ← DE − 1; HL ← HL − 1;
BC ← BC − 1

Format:

1	1	1	0	1	1	0	1	byte 1: ED
1	0	1	0	1	0	0	0	byte 2: A8

Description: The contents of the memory location addressed by HL are loaded into the memory location addressed by DE. Then BC, DE, and HL are all decremented.

Data Flow:



Timing: 4 M cycles; 16 T states; 8 usec @ 2 MHz

Addressing Modes: Indirect.

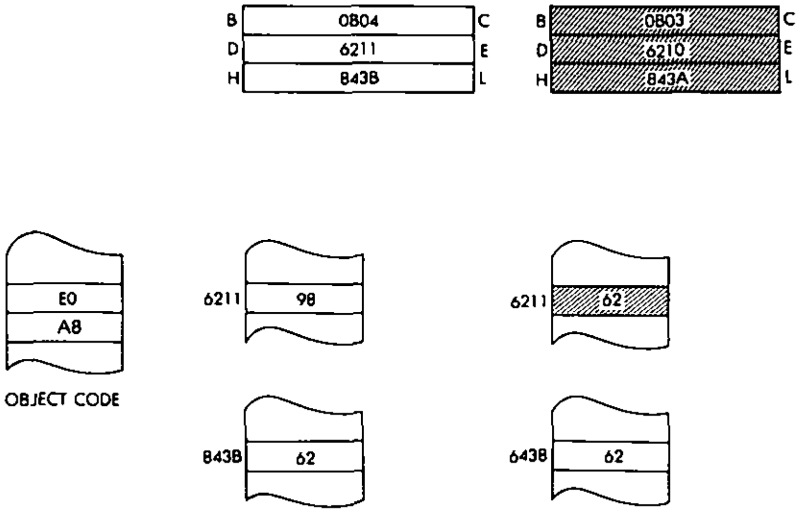
Flags:

S	Z		H		P/V	N	C
			O		X	O	

Reset if BC = 0 after execution, set otherwise.

Example: LDD

Before: After:



LDDR

Repeating block load with decrement.

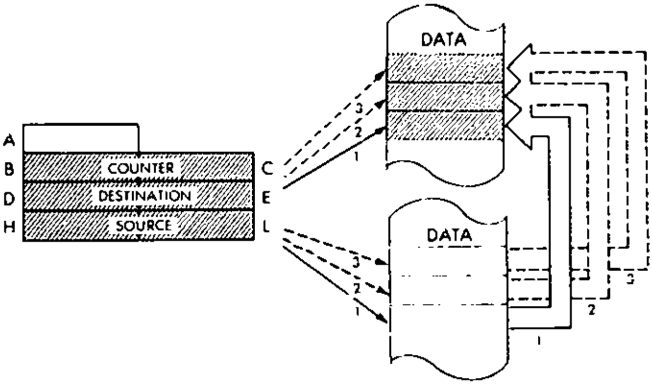
Function: (DE) ← (HL); DE ← DE − 1; HL ← HL − 1;
BC ← BC − 1; Repeat until BC = 0

Format:

1	1	1	0	1	1	0	1	byte 1: ED
1	0	1	1	1	0	0	0	byte 2: B8

Description: The contents of the memory location addressed by HL are loaded into the memory location addressed by DE. Then DE, HL, and BC are all decremented. If BC ≠ 0, then the program counter is decremented by 2 and the instruction re-executed.

Data Flow:



Timing: BC ≠ 0: 5 M cycles; 21 T states; 10.5 usec @ 2 MHz.
BC = 0: 4 M cycles; 16 T states; 8 usec @ 2 MHz

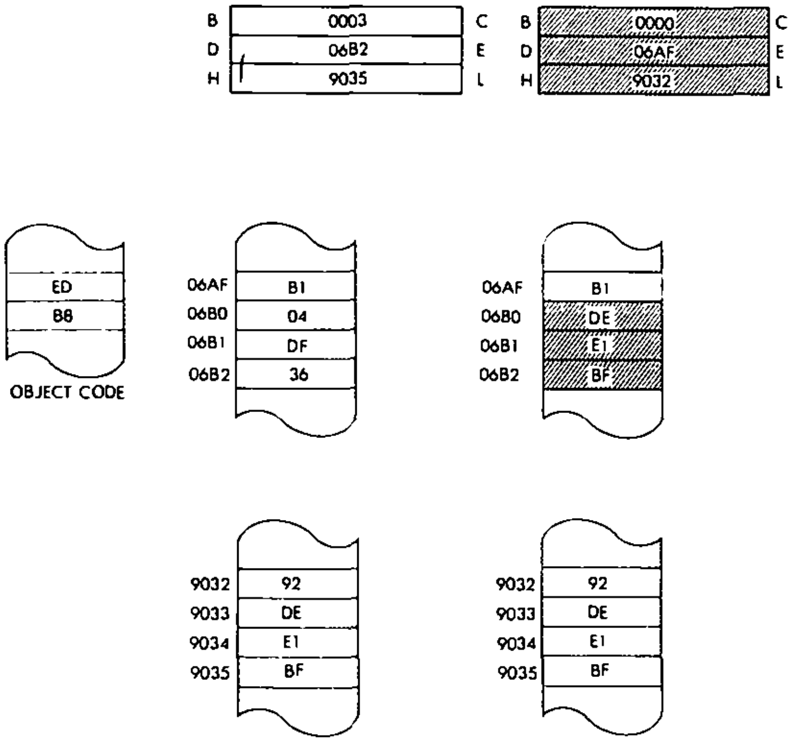
Addressing Mode: Indirect.

Flags:

S	Z		H		P/V	N	C
			○		○	○	

Example: LDDR

Before: After:



LDI

Block load with increment.

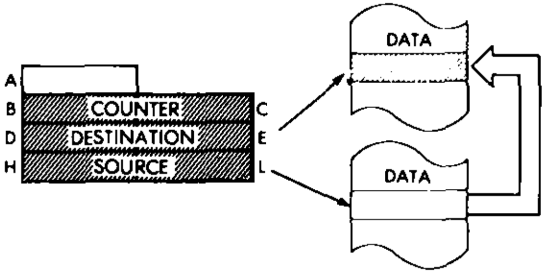
Function: (DE) ← (HL); DE ← DE + 1; HL ← HL + 1; BC ← BC − 1

Format:

1	1	1	0	1	1	0	1	byte 1: ED
1	0	1	0	0	0	0	0	byte 2: A0

Description: The contents of the memory location addressed by HL are loaded into the memory location addressed by DE. Then both DE and HL are incremented, and the register pair BC is decremented.

Data Flow:



Timing: 4 M cycles; 16 T states; 8 usec @ 2 MHz

Addressing Mode: Indirect.

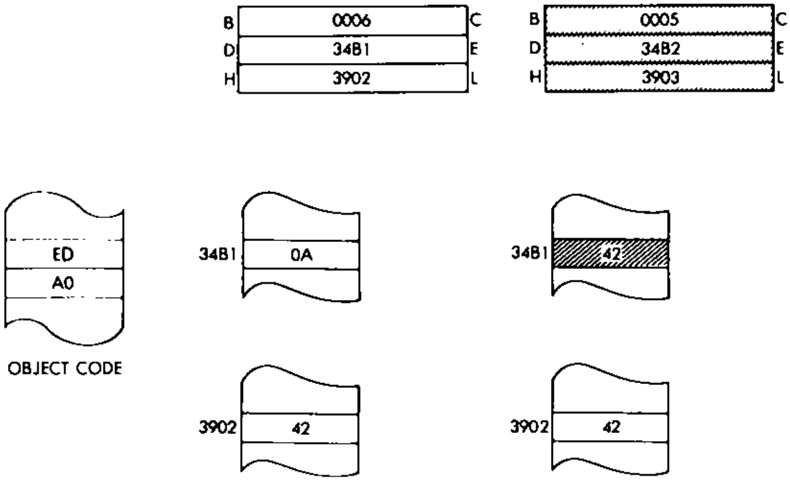
Flags:

S	Z		H		P/V	N	C
			O		X	O	

Reset if BC = 0 after execution, set otherwise.

Example: LDI

Before: After:



LDIR

Repeating block load with increment.

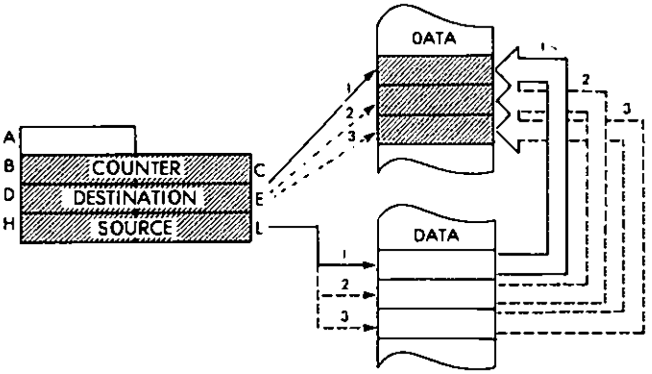
Function: (DE) ← (HL); DE ← DE + 1; HL ← HL + 1; BC ← BC − 1; Repeat until BC = 0

Format:

1	1	1	0	1	1	0	1	byte 1: ED
1	0	1	1	0	0	0	0	byte 2: B0

Description: The contents of the memory location addressed by HL are loaded into the memory location addressed by DE. Then both DE and HL are incremented. BC is decremented. If BC ≠ 0 then the program counter is decremented by 2 and the instruction is re-executed.

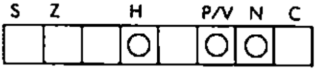
Data Flow:



Timing: For BC ≠ 0: 5M cycles; 21 T states; 10.5 usec @ 2 MHz.
For BC = 0: 4 M cycles; 16 T states; 8 usec @ 2 MHz

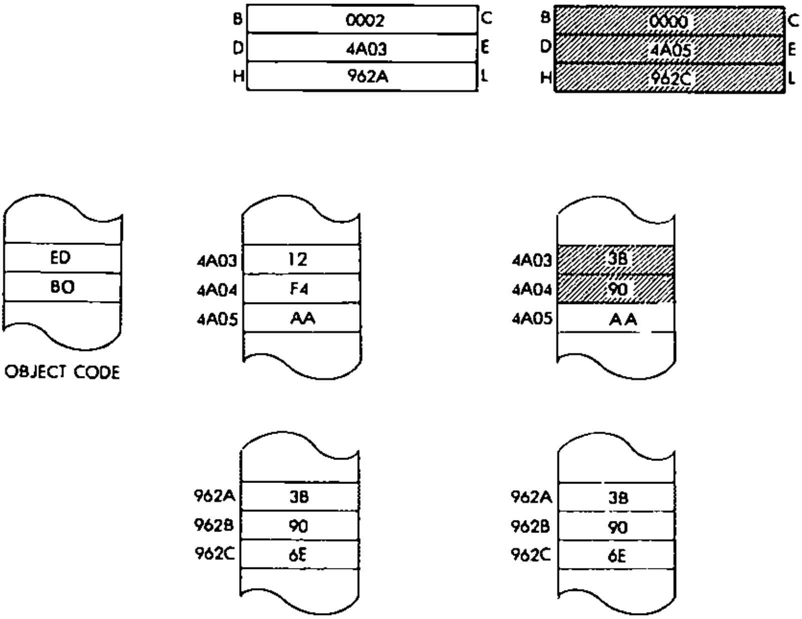
Addressing Mode: Indirect.

Flags:



Example: LDIR

Before: After:



LD r, (HL) Load register r indirect from memory location (HL).

Function: r ← (HL)

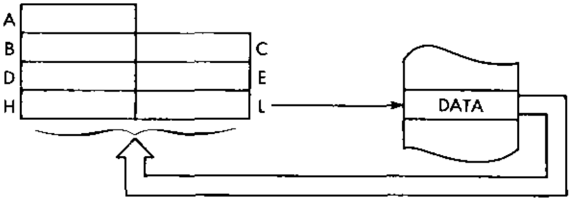
Format:

D	1	← r →	1	1	0

Description: The contents of the memory location addressed by HL are loaded into the specified register. r may be any one of:

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

Data Flow:



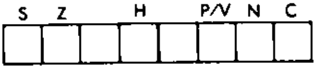
Timing: 2 M cycles; 7 T states; 3.5 usec @ 2 MHz

Addressing Mode: Indirect.

Byte Codes: r:

A	B	C	D	E	H	L
7E	46	4E	56	5E	66	6E

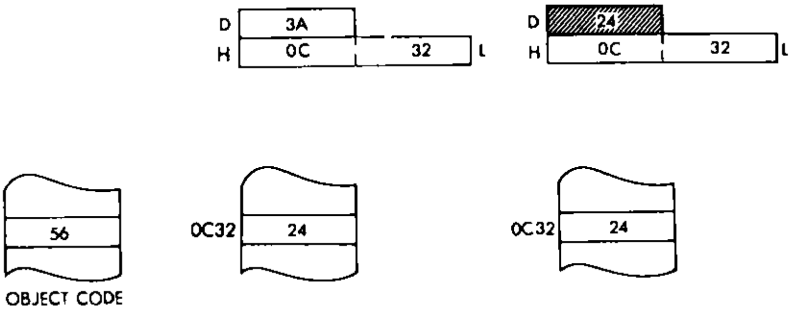
Flags:



(no effect).

Example: LD D, (HL)

Before: After:



NEG

Negate accumulator.

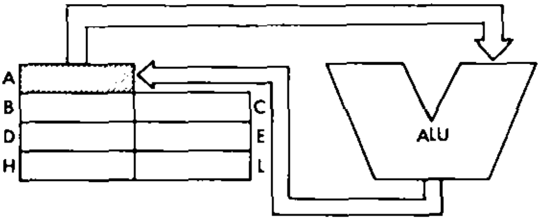
Function: A ← 0 − A

Format:

1	1	1	0	1	1	0	1	byte 1: ED
0	1	0	0	0	1	0	0	byte 2: 44

Description: The contents of the accumulator are subtracted from zero (two's complement) and the result is stored back in the accumulator.

Data Flow:



Timing: 2 M cycles; 8 T states; 4 usec @ 2 MHz

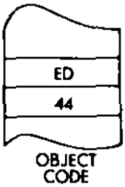
Addressing Mode: Implicit.

Flags:

S	Z		H		P/V	N	C
●	●		●		●	1	●

C will be set if A was 0 before the instruction. P will be set if A was 80H.

Example: NEG



Before:

A 32

After:

A CE

NOP No operation.

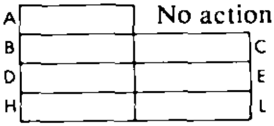
Function: Delay.

Format:

0	0	0	0	0	0	0	0	00

Description: Nothing is done for 1 M cycle.

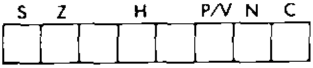
Data Flow:



Timing: 1 M cycle; 4 T states; 2 usec @ 2 MHz

Addressing Mode: Implicit

Flags:



(no effect).

OR s

Logical or accumulator and operand s.

Function: $A \leftarrow A \vee s$

Format: s: may be r, n, (HL), (IX+ d), or (IY + d)

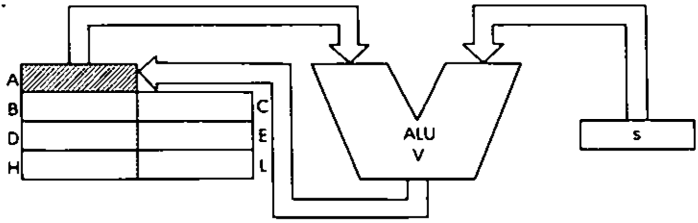
r	1	0	1	1	0	← r →			
n	1	1	1	1	0	1	1	0	byte 1: F6
	← n →								byte 2: immediate data
(HL)	1	0	1	1	0	1	1	0	byte 1: B6
(IX + d)	1	1	0	1	1	1	0	1	byte 1: DD
	1	0	1	1	0	1	1	0	byte 2: B6
	← d →								byte 3: offset value
(IY + d)	1	1	1	1	1	1	0	1	byte 1: FD
	1	0	1	1	0	1	1	0	byte 2: B6
	← d →								byte 3: offset value

r may be any one of:

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

Description: The accumulator and the specified operand are logically 'or'ed, and the result is stored in the accumulator. s is defined in the description of the similar ADD instructions.

Data Flow:



Timing:

s:	*M cycles:*	*T states:*	*usec* *@ 2 MHz:*
r	1	4	4
n	2	7	3.5
(HL)	2	7	3.5
(IX + d)	5	19	9.5
(IY + d)	5	19	9.5

Addressing Mode: r: implicit; n: immediate; (HL): indirect; (IX + d), (IY + d): indexed.

Byte Codes: OR r

r:	A	B	C	D	E	H	L
	B7	B0	B1	B2	B3	B4	B5

Flags:

S	Z		H		P/V	N	C
●	●		○		●	○	○

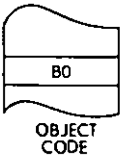
Example: OR B

Before:

A	06
B	B9

After:

A	BF
B	B9



OTDR

Block output with decrement

Function: (C)←(HL); B←B − 1; HL←HL − 1;
Repeat until B = 0.

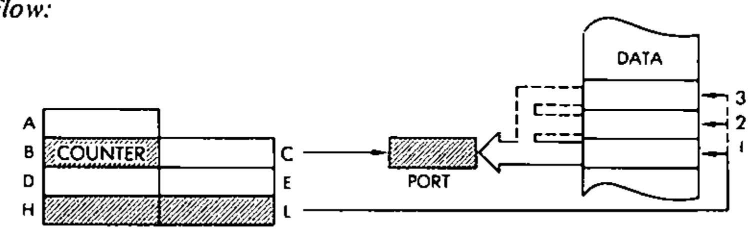
Format:

1	1	1	0	1	1	0	1	byte 1: ED

1	0	1	1	1	0	1	1	byte 2: BB

Description: The contents of the memory location addressed by the HL register pair are output to the peripheral device addressed by the contents of the C register. Both the B register and the HL register pair are then decremented. If B ≠ 0, the program counter is decremented by 2 and the instruction is re-executed. C supplies bits A0 to A7 of the address bus. B supplies (after decrementation) bits A8 to A15.

Data Flow:



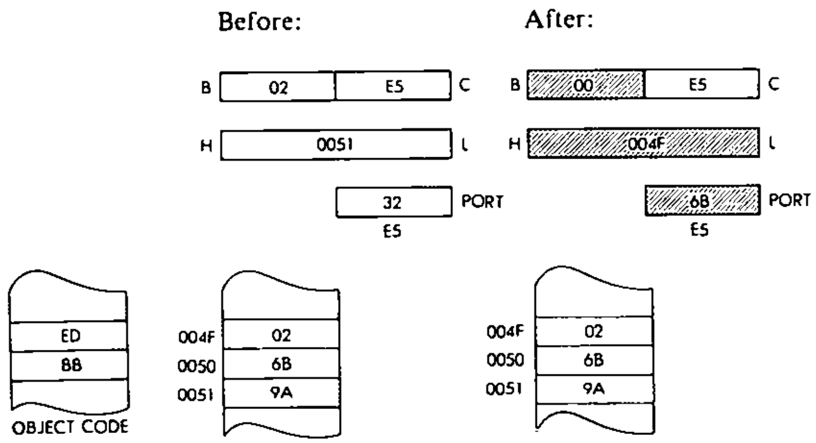
Timing: B = 0: 4 M cycles; 16 T states; 8 usec @ 2 MHz.
B ≠ 0: 5 M cycles; 21 T states; 10.5 usec @ 2 MHz

Addressing Mode: External.

Flags:

S	Z		H		P/V	N	C
?	1		?		?	1	

Example: OTDR



OTIR

Block output with increment.

Function: (C) ← (HL); B ← B − 1; HL ← HL + 1; Repeat until B = 0

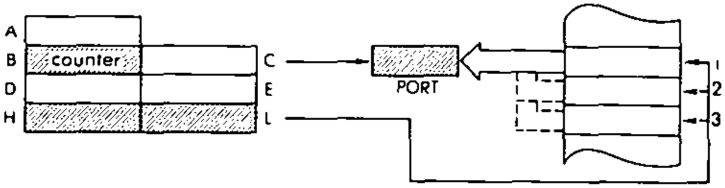
Format:

1	1	1	0	1	1	0	1	byte 1: ED

1	0	1	1	0	0	1	1	byte 2: B3

Description: The contents of the memory location addressed by the HL register pair are output to the peripheral device addressed by the contents of the C register. The B register is decremented and the HL register pair is incremented. If B ≠ 0, the program counter is decremented by 2 and the instruction is re-executed. C supplies bits A0 to A7 of the address bus. B supplies (after decrementation) bits A8 to A15.

Data Flow:



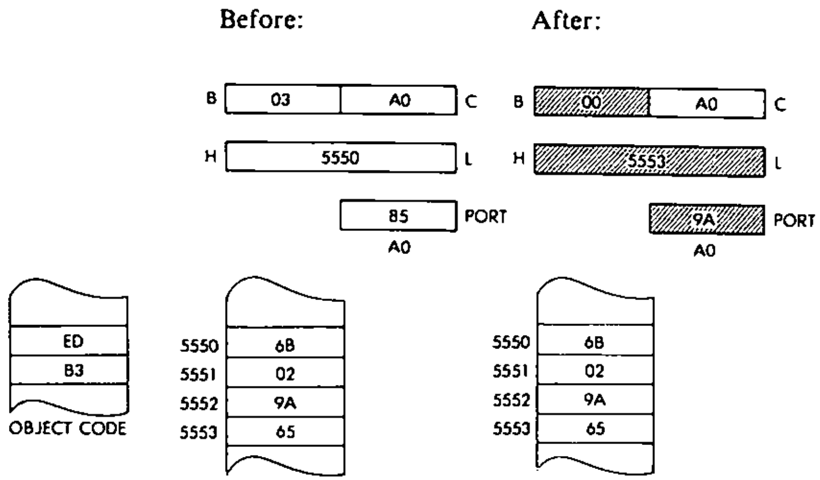
Timing: B = 0: 4 M cycles; 16 T states; 8 usec @ 2 MHz.
B ≠ 0: 5 M cycles; 21 T states; 10.5 usec @ 2 MHz

Addressing Mode: External.

Flags:

S	Z		H		P/V	N	C
?	1		?		?	1	

Example: OTIR



OUT (C), r

Output register r to port C.

Function: (C) ← r

Format:

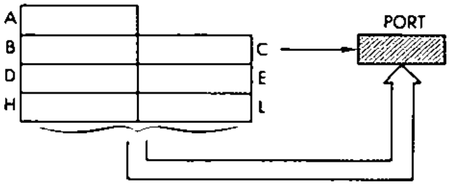
1	1	1	0	1	1	0	1	byte 1: ED
0	1	←	r	→	0	0	1	byte 2

Description: The contents of the specified register are output to the peripheral device addressed by the contents of the C register. r may be any one of:

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

Register C supplies bits A0 to A7 of the address bus. Register B supplies bits A8 to A15.

Data Flow:



Timing: 3 M cycles; 12 T states; 6 usec @ 2 MHz

Addressing Mode: External.

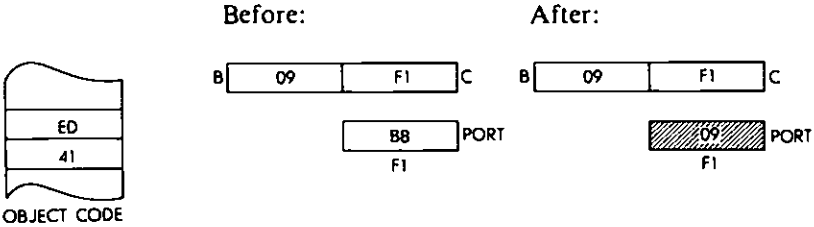
Flags:

S	Z		H		P/V	N	C	
								(no effect).

Byte Codes:

	A	B	C	D	E	H	L
ED-	79	41	49	51	59	61	69

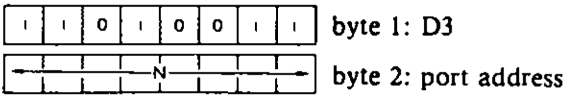
Example: OUT (C), B



OUT (N), A Output accumulator to peripheral port N.

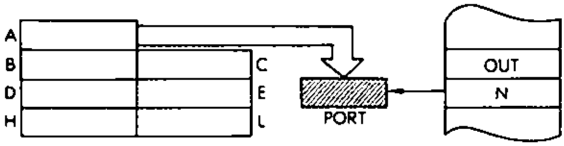
Function: (N) ← A

Format:



Description: The contents of the accumulator are output to the peripheral device addressed by the contents of the memory location immediately following the op-code.

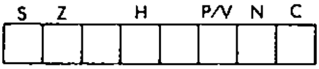
Data Flow:



Timing: 3 M cycles, 11 T states; 5.5 usec @ 2 MHz

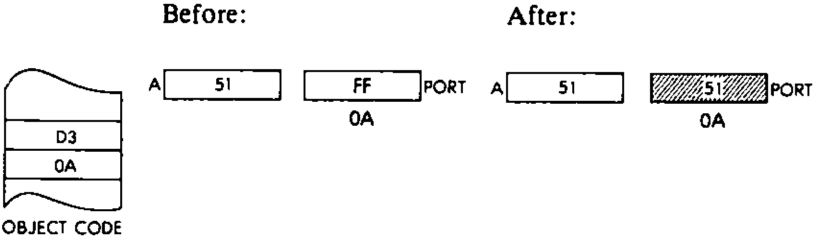
Addressing Mode: External.

Flags:



(no effect).

Example: OUT (0A), A



OUTD

Output with decrement.

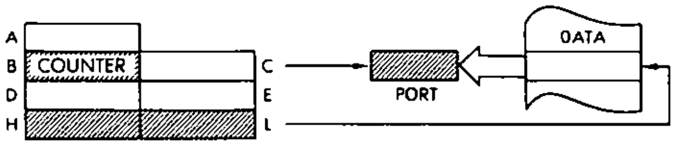
Function: (C) ← (HL); BC ← B − 1; HL ← HL − 1

Format:

1	1	1	0	1	1	0	1	byte 1: ED
1	0	1	0	1	0	1	1	byte 2: AB

Description: The contents of the memory location addressed by the HL register pair are output to the peripheral device addressed by the contents of the C register. Then both the B register and the HL register pair are decremented. C supplies bits A0 to A7 of the address bus. B supplies (after decrementation) A8 to A15.

Data Flow:



Timing: 4 M cycles; 16 T states; 8 usec @ 2 MHz

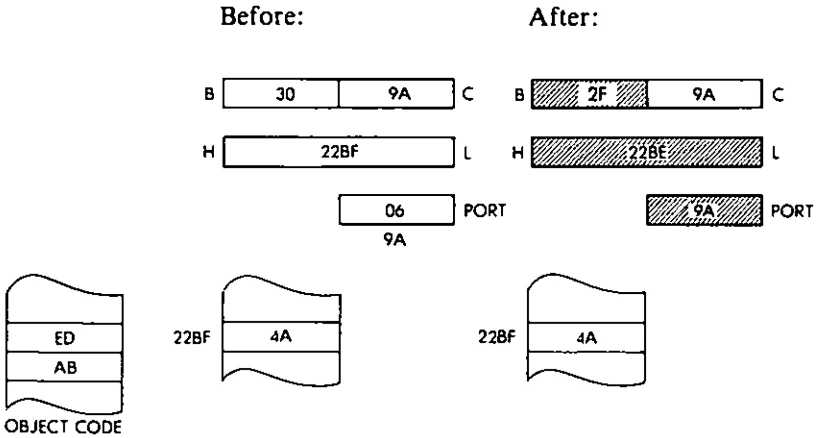
Addressing Mode: External.

Flags:

S	Z		H		P/V	N	C
?	X		?		?	1	

Set if B = 0 after execution, reset otherwise.

Example: OUTD



OUTI

Output with increment.

Function: (C) ← (HL); B ← B − 1; HL ← HL + 1

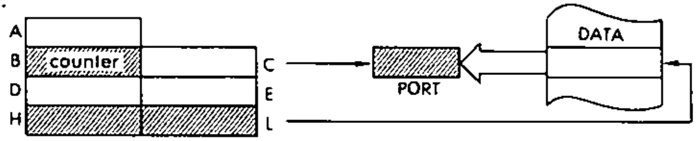
Format:

1	1	1	0	1	1	0	1	byte 1: ED
1	0	1	0	0	0	1	1	byte 2: A3

Description: The contents of the memory location addressed by the HL register pair are output to the peripheral device addressed by the C register. The B register is decremented and the HL register pair is incremented.

C supplies bits A0 to A7 of the address bus.
B (after decrementation) supplies bits A8 to A15.

Data Flow:



Timing: 4 M cycles; 16 T states; 8 usec @ 2 MHz

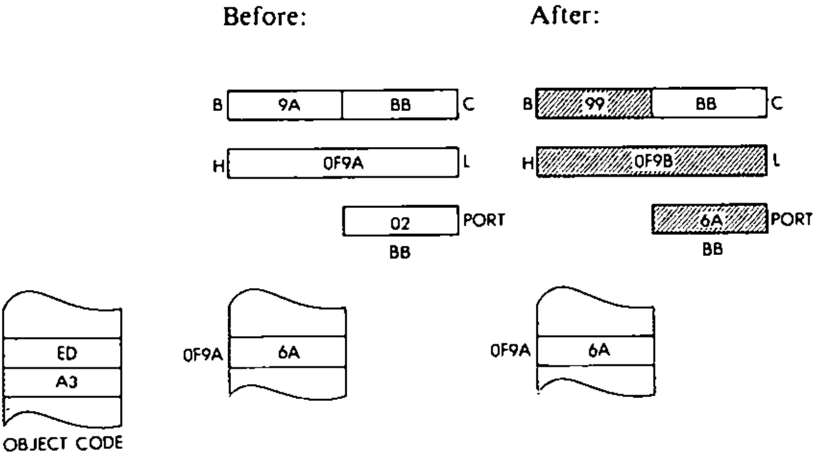
Addressing Mode: External.

Flags:

S	Z		H		P/V	N	C
?	X		?		?	1	

Z: Set if B = 0 after execution, reset otherwise.

Example: OUTI



POP qq

Pop register pair qq from stack.

Function: $qq_{low} \leftarrow (SP); qq_{high} \leftarrow (SP + 1); SP \leftarrow SP + 2$

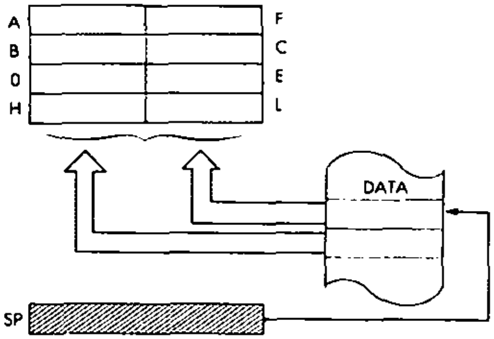
Format:

1	1	q	q	0	0	0	1

Description: The contents of the memory location addressed by the stack pointer are loaded into the low order of the specified register pair and then the stack pointer is incremented. The contents of the memory location now addressed by the stack pointer are loaded into the high order of the register pair, and the stack pointer is again incremented. qq may be any one of:

BC	00	HL	10
DE	01	AF	11

Data Flow:



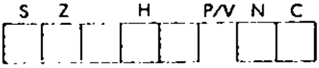
Timing: 3 M cycles; 10 T states; 5 usec @ 2 MHz

Addressing Mode: Indirect.

Byte Codes:

qq:	BC	DE	HL	AF
	C1	D1	E1	F1

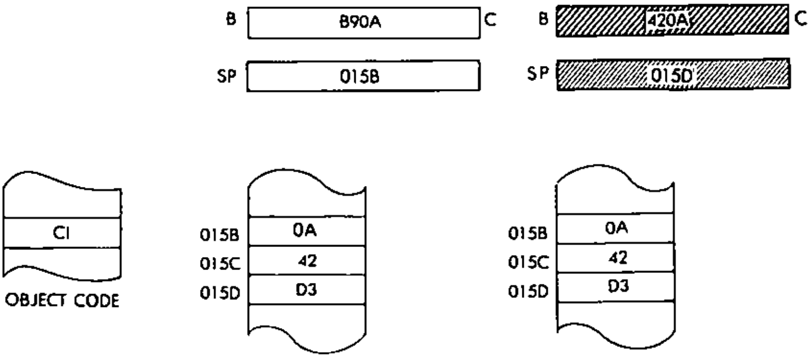
Flags:



(no effect).

Example: POP BC

Before: After:



POP IX

POP IX register from stack.

Function: $IX_{low} \leftarrow (SP);\ IX_{high} \leftarrow (SP + 1);\ SP \leftarrow SP + 2$

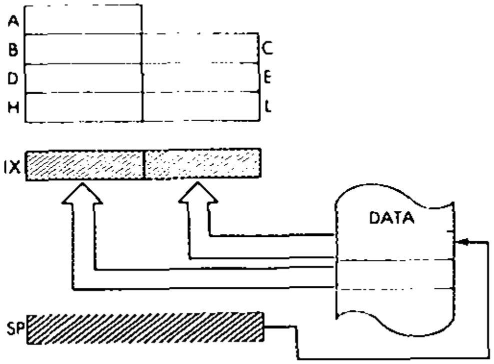
Format:

1	1	0	1	1	1	0	1	byte 1: DD

1	1	1	0	0	0	0	1	byte 2: E1

Description: The contents of the memory location addressed by the stack pointer are loaded into the low order of the IX register, and the stack pointer is incremented. The contents of the memory location now addressed by the stack pointer are loaded into the high order of the IX register, and the stack pointer is again incremented.

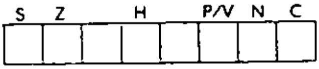
Data Flow:



Timing: 4 M cycles; 14 T states; 7 usec @ 2 MHz

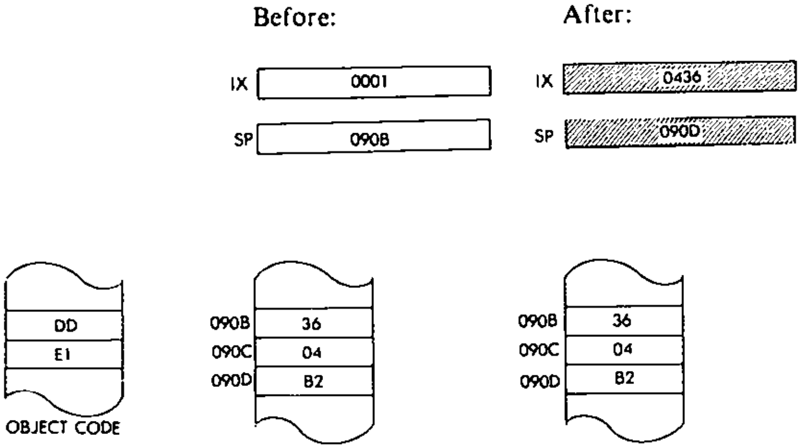
Addressing Mode: Indirect.

Flags:



(no effect).

Example: POP IX



POP IY

POP IY register from stack.

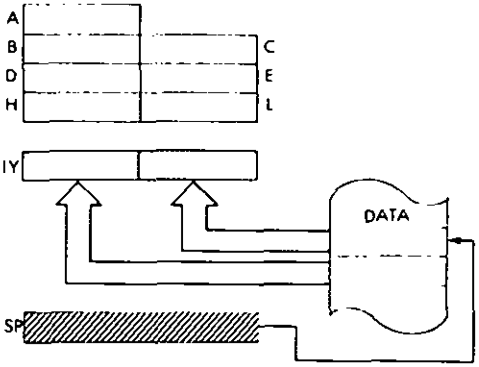
Function: $IY_{low} \leftarrow (SP); IY_{high} \leftarrow (SP + 1); SP \leftarrow SP + 2$

Format:

1	1	1	1	1	1	0	1	byte 1: FD
1	1	1	0	0	0	0	1	byte 2: E1

Description: The contents of the memory location addressed by the stack pointer are loaded into the low order of the IY register, and then the stack pointer is incremented. The contents of the memory location now addressed by the stack pointer are loaded into the high order of the IY register, and the stack pointer is again incremented.

Data Flow:



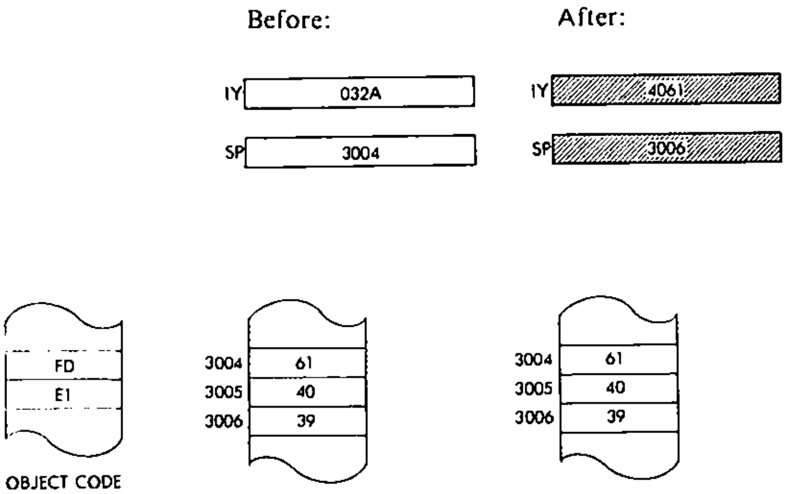
Timing: 4 M cycles; 14 T states; 2 usec @ 2 MHz

Addressing Mode: Indirect.

Flags:

S	Z		H		P/V	N	C	
								(no effect).

Example: POP IY



PUSH qq

Push register pair onto stack.

Function: $(SP - 1) \leftarrow qq_{high}; (SP - 2) \leftarrow qq_{low};$
$SP \leftarrow SP - 2$

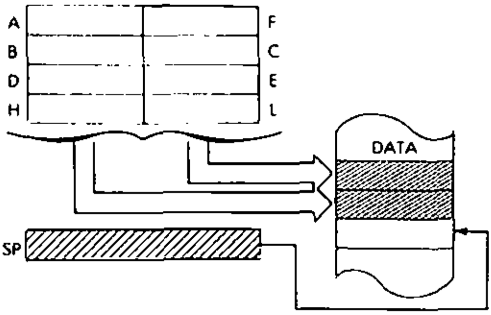
Format:

1	1	q	q	0	1	0	1

Description: The stack pointer is decremented and the contents of the high order of the specified register pair are then loaded into the memory location addressed by the stack pointer. The stack pointer is again decremented and the contents of the low order of the register pair are loaded into the memory location currently addressed by the stack pointer. qq may be any one of:

BC − 00	HL − 10
DE − 01	AF − 11

Data Flow:



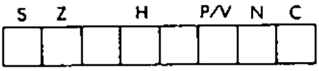
Timing: 3 M cycles; 11 T states; 6.5 usec @ 2 MHz

Addressing Mode: Indirect.

Byte Codes:

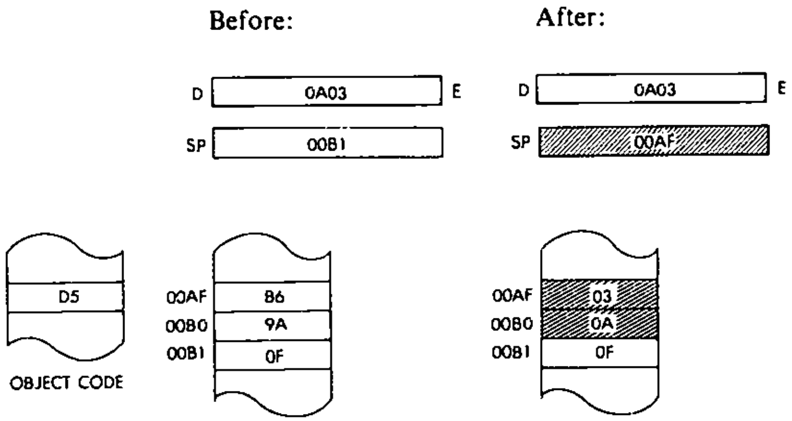
qq:	BC	DE	HL	AF
	C5	D5	E5	F5

Flags:



(no effect).

Example: PUSH DE



PUSH IX

Push IX onto stack.

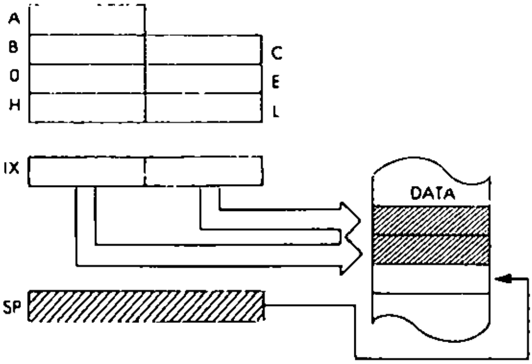
Function: $(SP - 1) \leftarrow IX_{high}; (SP - 2) \leftarrow IX_{low};$
$SP \leftarrow SP - 2$

Format:

1	1	0	1	1	1	0	1	byte 1: DD
1	1	1	0	0	1	0	1	byte 2: E5

Description: The stack pointer is decremented, and the contents of the high order of the IX register are loaded into the memory location addressed by the stack pointer. The stack pointer is again decremented and then the contents of the low order of the IX register are loaded into the memory location addressed by the stack pointer.

Data Flow:



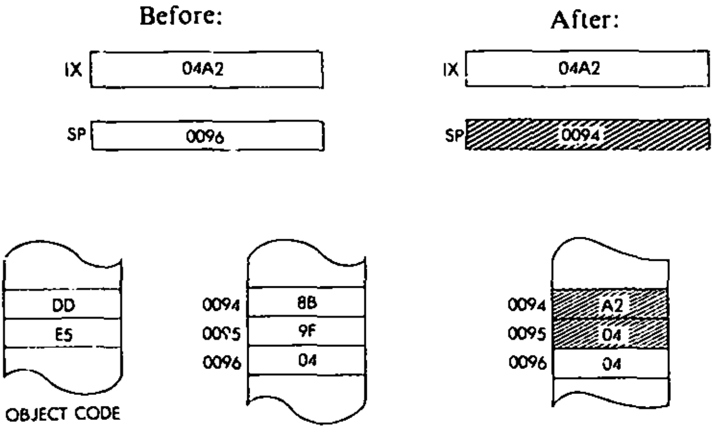
Timing: 4 M cycles; 15 T states; 7.5 usec @ 2 MHz

Addressing Mode: Indirect.

Flags:

S	Z		H		P/V	N	C	
□	□	□	□	□	□	□	□	(no effect)

Example: PUSH IX



PUSH IY

Push IY onto stack.

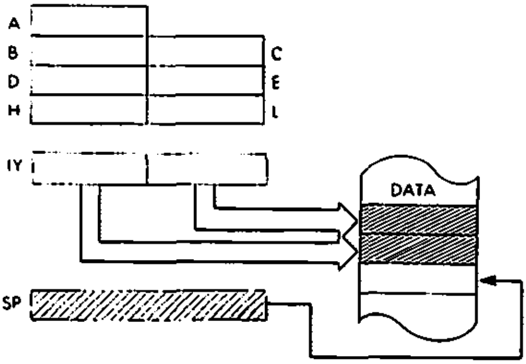
Function: $(SP - 1) \leftarrow IY_{high}$; $(SP - 2) \leftarrow IY_{low}$;
$SP \leftarrow SP - 2$

Format:

1	1	1	1	1	1	0	1	byte 1: FD
1	1	1	0	0	1	0	1	byte 2: E5

Description: The stack pointer is decremented and the contents of the high order of the IY register are loaded into the memory location addressed by the stack pointer. The stack pointer is again decremented and the contents of the low order of the IY register are loaded into the memory location addressed by the stack pointer.

Data Flow:



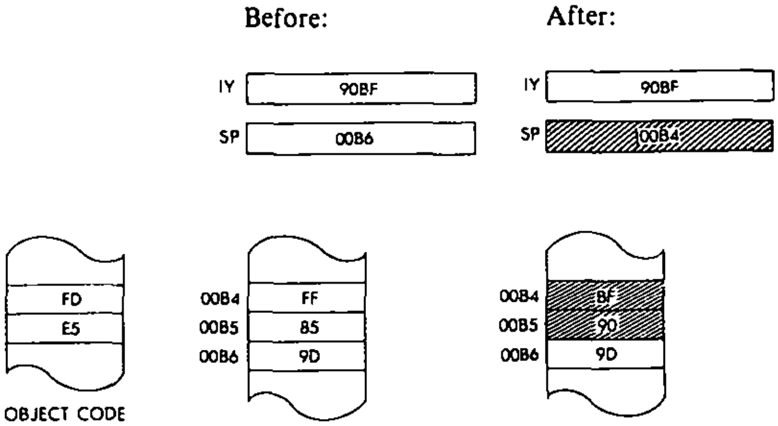
Timing: 3 M cycles; 15 T states; 7.5 usec @ 2 MHz

Addressing Mode: Indirect.

Flags:

S	Z		H		P/V	N	C	
								(no effect)

Example: PUSH IY



RES b, s

Reset bit b of operand s.

Function: $s_b \leftarrow 0$

Format:

s:									
r	1	1	0	0	1	0	1	1	byte 1: CB
	1	0	←	b	→	←	r	→	byte 2
(HL)	1	1	0	0	1	0	1	1	byte 1: CB
	1	0	←	b	→	1	1	0	byte 2
(IX + d)	1	1	0	1	1	1	0	1	byte 1: DD
	1	1	0	0	1	0	1	1	byte 2: CB
	←			d				→	byte 3: offset value
	1	0	←	b	→	1	1	0	byte 4
(IY + d)	1	1	1	1	1	1	0	1	byte 1: FD
	1	1	0	0	1	0	1	1	byte 2: CB
	←			d				→	byte 3: offset value
	1	0	←	b	→	1	1	0	byte 4

b may be any one of:

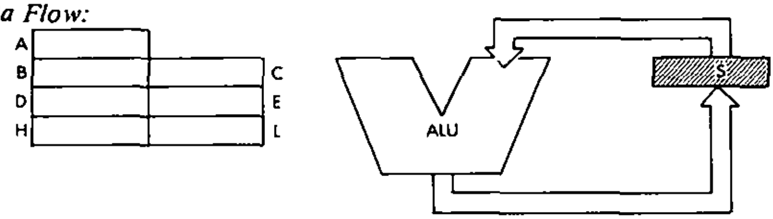
0 – 000	4 – 100
1 – 001	5 – 101
2 – 010	6 – 110
3 – 011	7 – 111

r may be any one of:

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

Description: The specified bit of the location determined by s is reset. s is defined in the description of the similar BIT instructions.

Data Flow:



Timing:

s:	*M cycles:*	*T states:*	*usec @ 2 MHz:*
r	2	8	4
(HL)	4	15	7.5
(IX + d)	6	23	11.5
(IY + d)	6	23	11.5

Addressing Mode: r: implicit; (HL): indirect; (IX + d), (IY + d): indexed.

Byte Codes: RES b, r

	b: \ r:	A	B	C	D	E	H	L
CB—	0	87	80	81	82	83	84	85
	1	8F	88	89	8A	88	8C	8D
	2	97	90	91	92	93	94	95
	3	9F	98	99	9A	9B	9C	9D
	4	A7	A0	A1	A2	A3	A4	A5
	5	AF	A8	A9	AA	A8	AC	AD
	6	B7	80	B1	82	B3	B4	B5
	7	BF	88	B9	BA	BB	8C	8D

RES b, (HL)

b:	0	1	2	3	4	5	6	7
CB—	86	8E	96	9E	A6	AE	86	BE

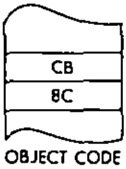
RES b, (IX + d) DDCB –
RES r/ (HL) CB –
RES b, (IY + d) FDCB –

b:	0	1	2	3	4	5	6	7
	86	8E	96	9E	A6	AE	B6	BE

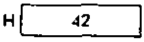
Flags:

S	Z		H		P/V	N	C	
								(No effect)

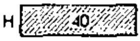
Examples: RES 1, H



Before:



After:



RET

Return from subroutine

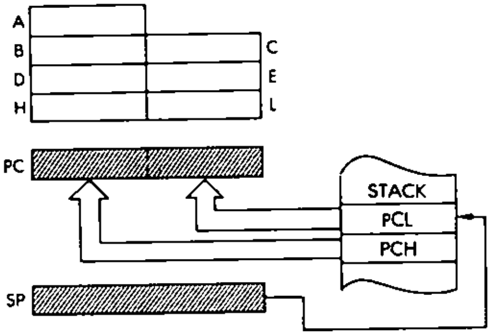
Function: $PC_{low} \leftarrow (SP); PC_{high} \leftarrow (SP + 1); SP \leftarrow SP + 2$

Format:

1	1	0	0	1	0	0	1	C9

Description: The program counter is popped off the stack as described for the POP instructions. The next instruction fetched is from the location pointed to by PC.

Data Flow:



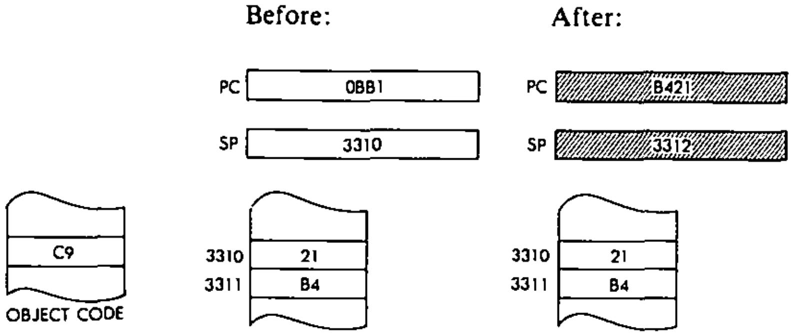
Timing: 3 M cycles; 10 T states; 5 usec @ 2 MHz

Addressing Mode: Indirect.

Flags:

S	Z		H		P/V	N	C	
								(no effect)

Example: RET


Before:
After:
PC
0BB1
SP
3310
PC
B421
SP
3312
C9
OBJECT CODE
3310
21
3311
B4
3310
21
3311
B4

RET cc

Return from subroutine on condition.

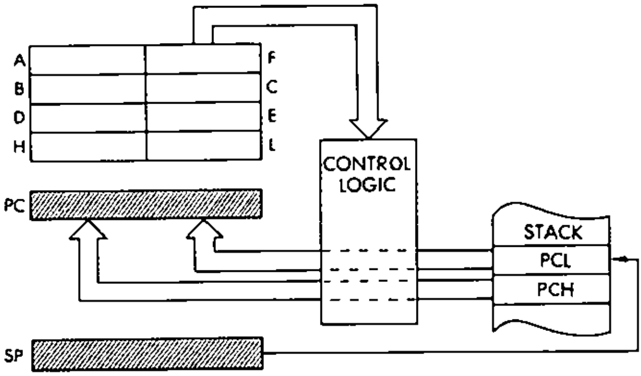
Function: If cc true: $PC_{low} \leftarrow (SP)$; $PC_{high} \leftarrow (SP + 1)$; $SP \leftarrow SP + 2$

Format:

1	1	←	cc	→	0	0	0

Description: If the condition is met, the contents of the program counter are popped off the stack as described for the POP instructions. The next instruction is fetched from the address in PC. If the condition is not met, instruction execution continues in sequence.

Data Flow:



cc may be any one of:

NZ	000	PO	100
Z	001	PE	101
NC	010	P	110
C	011	M	111

Timing: Condition met: 3 M cycles; 11 T states; 6.5 usec @ 2 MHz.
Condition not met: 1 M cycle; 5 T states; 2.5 usec @ 2 MHz

Addressing Mode: Indirect.

Byte Codes:

CC.	NZ	Z	NC	C	PO	PE	P	M
	C0	C8	D0	D8	E0	E8	F0	F8

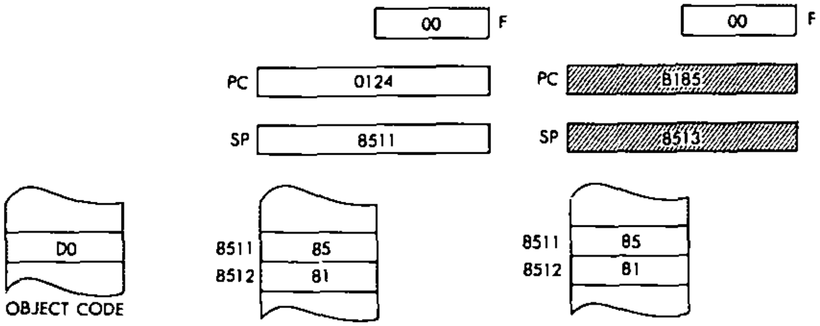
Flags:

S	Z		H		P/V	N	C

(no effect)

Example: RET NC

Before: After:



RETI

Return from interrupt.

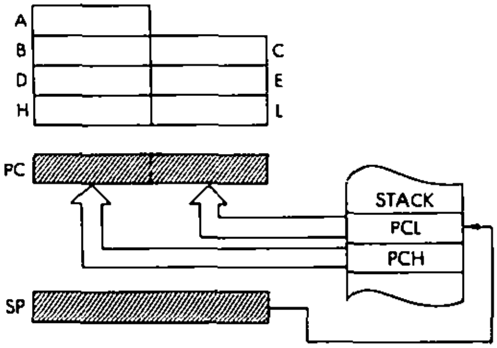
Function: $PC_{low} \leftarrow (SP); PC_{high} \leftarrow (SP + 1); SP \leftarrow SP + 2$

Format:

1	1	1	0	1	1	0	1	byte 1: ED
0	1	0	0	1	1	0	1	byte 2: 4D

Description: The program counter is popped off the stack as described for the POP instructions. This instruction is recognized by Zilog peripheral devices as the end of a peripheral service routine so as to allow proper control of nested priority interrupts. An EI instruction must be executed prior to RETI in order to re-enable interrupts.

Data Flow:



Timing: 4 M cycles; 14 T states; 7 usec @ 2 MHz

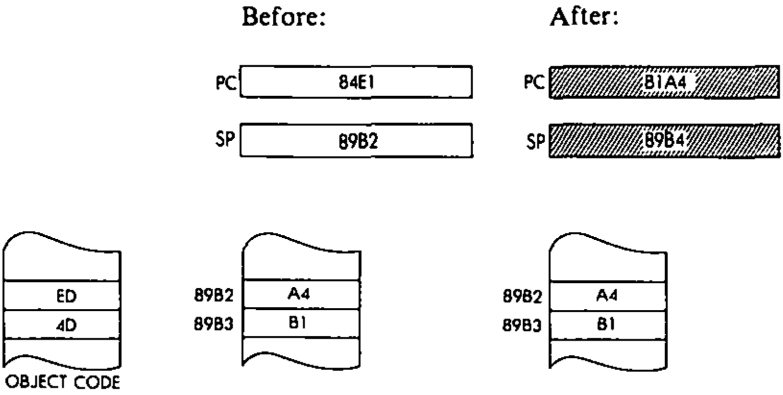
Addressing Modes: *Indirect.*

Flags:

S	Z		H		P/V	N	C

(no effect).

Example: RETI



RETN

Return from non-maskable interrupt.

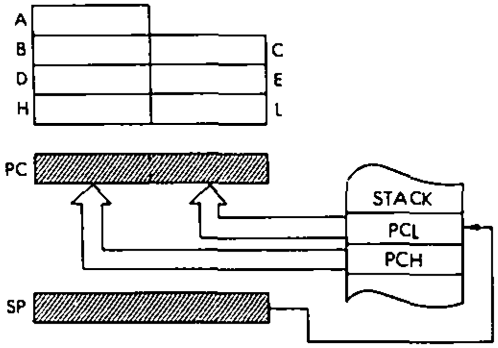
Function: $PC_{low} \leftarrow (SP)$; $PC_{high} \leftarrow (SP + 1)$; $SP \leftarrow SP + 2$; IFF'I ← IFF2

Format:

1	1	1	0	1	1	0	1	byte 1: ED
0	1	0	0	0	1	0	1	byte 2: 45

Description: The program counter is popped off the stack as described for the POP instructions. Then the contents of the IFF2 (storage flip-flop) is copied back into the IFF1 to restore the state of the interrupt flag before the non-maskable interrupt.

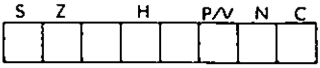
Data Flow:



Timing: 4 M cycles; 14 T states; 7 usec @ 2 MHz

Addressing Mode: Indirect.

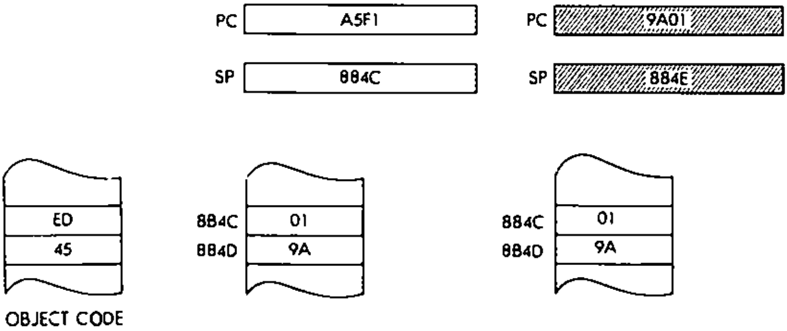
Flags:



(no effect).

Example: RETN

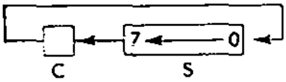
Before: After:



RL s

Rotate left through carry operand s.

Function:



Format: *s:*

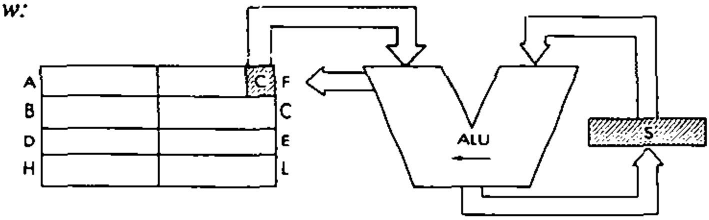
r	1	1	0	0	1	0	1	1	byte 1: CB
	0	0	0	1	0	←r→			byte 2
(HL)	1	1	0	0	1	0	1	1	byte 1: CB
	0	0	0	1	0	1	1	0	byte 2: 16
(IX + d)	1	1	0	1	1	1	0	1	byte 1: DD
	1	1	0	0	1	0	1	1	byte 2: CB
	← d →								byte 3: offset value
	0	0	0	1	0	1	1	0	byte 4: 16
(IY + d)	1	1	1	1	1	1	0	1	byte 1: FD
	1	1	0	0	1	0	1	1	byte 2: CB
	← d →								byte 3: offset value
	0	0	0	1	0	1	1	0	byte 4: 16

r may be any one of:

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

Description: The contents of the location of the specific operand are shifted left one bit place. The contents of the carry flag are moved to bit 0 and the contents of bit 7 are moved to the carry flag. The final result is stored back in the original location. s is defined in the description of the similar RLC instructions.

Data Flow:



Timing:

s:	*M cycles:*	*T states:*	*usec @ 2 MHz:*
r	2	8	4
(HL)	4	15	7.5
(IX + d)	6	23	11.5
(IY + d)	6	23	11.5

Addressing Mode: r: implicit; (HL): indirect; (IX + d), (IY + d): indexed.

Byte Codes: RL r

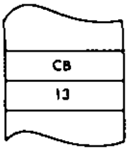
r:	A	B	C	D	E	H	L
CB-	17	10	11	12	13	14	15

Flags:

S	Z		H		Ⓟ/V	N	C
●	●		○		●	○	●

C is set by bit 7 of source.

Example: RL E



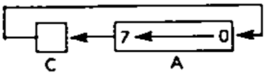
OBJECT CODE

Before:		After:	
41	F	B4	F
6E	E	DD	E

RLA

Rotate accumulator left through carry flag.

Function:

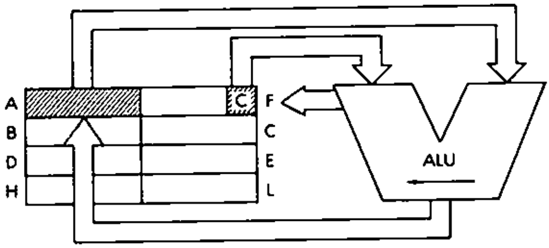


Format:

0	0	0	1	0	1	1	1	17

Description: The contents of the accumulator are shifted left one bit position. The contents of the carry flag are moved into bit 0 and the original contents of bit 7 are moved into the carry flag. (9 bit rotation.)

Data Flow:



Timing: 1 M cycle; 4 T states; 2 usec @ 2 MHz

Addressing Mode: Implicit.

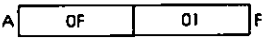
Flags:

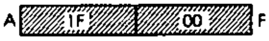
S	Z		H		P/V	N	C
			○			○	●

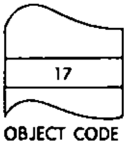
C is set by bit 7 of A.

Example: RLA

Before: After:



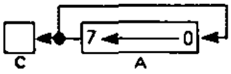




RLCA

Rotate accumulator left with branch carry.

Function:



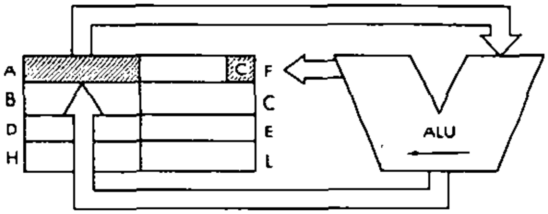
Format:

0	0	0	0	0	1	1	1

07

Description: The contents of the accumulator are rotated left one bit position. The original contents of bit 7 is moved to the carry flag as well as to bit 0.

Data Flow:



Timing: 1 M cycle; 4 T states; 2 usec @ 2 MHz

Addressing Mode: Implicit.

Flags:

S	Z		H		P/V	N	C
			○			○	●

C is set by bit 7 of A.

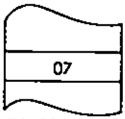
Example: RLCA

Before:

A	6B	01	F

After:

A	D6	00	F



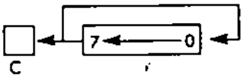
OBJECT CODE

Note: This instruction is identical to RLC A, except for the flags. It is provided for compatibility with the 8080.

RLC r

Rotate register r left with branch carry.

Function:



Format:

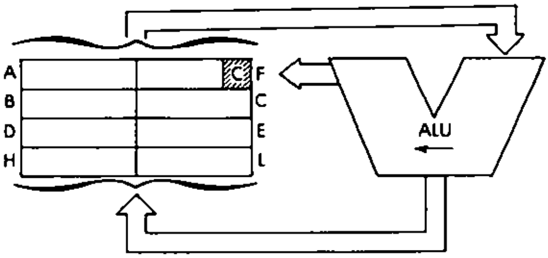
1	1	0	0	1	0	1	1	byte 1: CB

0	0	0	0	0	← r →			byte 2

Description: The contents of the specified register are rotated left. The original contents of bit 7 are moved to the carry flag as well as bit 0. r may be any one of:

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

Data Flow:



Timing: 2 M cycles; 8 T states; 4 usec @ 2 MHz

Addressing Mode: Implicit.

Byte Codes: r:

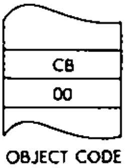
	A	B	C	D	E	H	L
CB-	07	00	01	02	03	04	05

Flags:

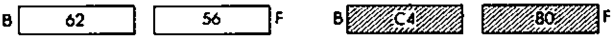
S	Z		H		Ⓟ/V	N	C
●	●		○		●	○	●

C is set by bit 7 of source register.

Example: RLC B



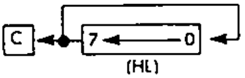
Before: After:



RLC (HL)

Rotate left with branch carry memory location (HL).

Function:

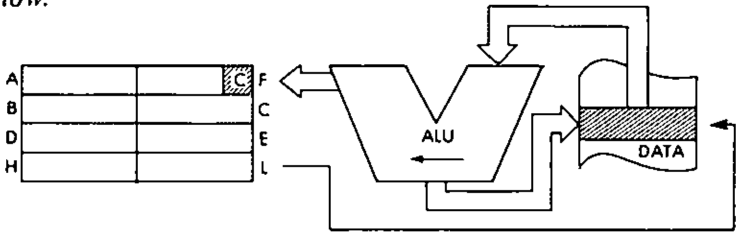


Format:

1	1	0	0	1	0	1	1	byte 1: CB
0	0	0	0	0	1	1	0	byte 2: 06

Description: The contents of the memory location addressed by the contents of the HL register pair are rotated left one bit position and the result is stored back at that location. The contents of bit 7 are moved to the carry flag as well as to bit 0.

Data Flow:



Timing: 4 M cycles; 15 T states; 7.5 usec @ 2 MHz

Addressing Mode: Indirect.

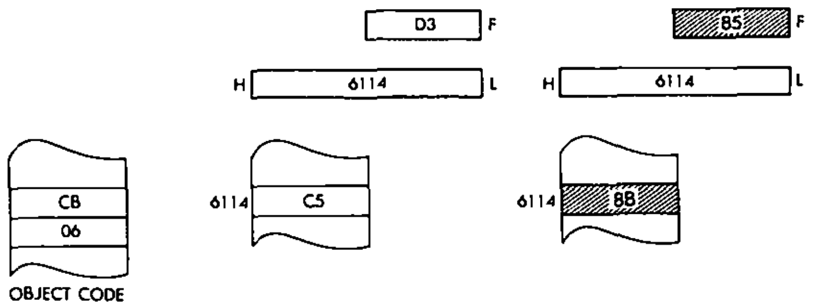
Flags:

S	Z		H		Ⓟ/V	N	C
●	●		○		●	○	●

C is set by bit 7 of the memory location.

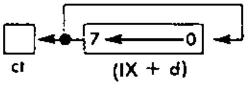
Example: RLC (HL)

Before: After:



RLC (IX + d) Rotate left with branch carry memory location (IX + d)

Function:

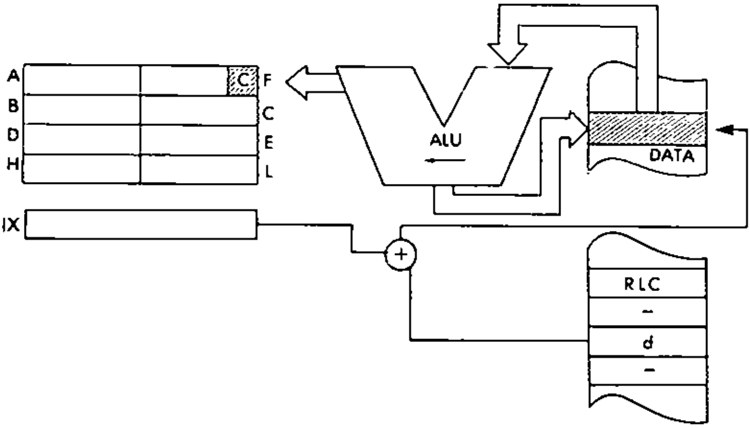


Format:

1	1	0	1	1	1	0	1	byte 1: DD
1	1	0	0	1	0	1	1	byte 2: CB
d								byte 3: offset value
0	0	0	0	0	1	1	0	byte 4: 06

Description: The contents of the memory location addressed by the contents of the IX register plus the given offset value are rotated left and the result is stored back at that location. The contents of bit 7 are moved to the carry flag as well as to bit 0.

Data Flow:



Timing: 6 M cycles; 23 T states; 11.5 usec @ 2 MHz

Addressing Mode: Indexed.

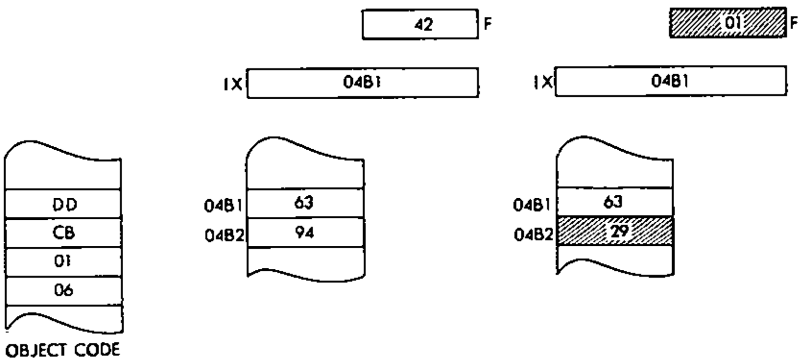
Flags:

S	Z		H		P/V	N	C
●	●		○		●	○	●

C is set by bit 7 of memory location.

Example: RLC (IX + 1)

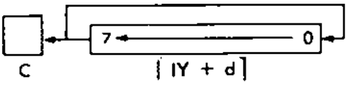
Before: After:



RLC (IY + d)

Rotate left with carry memory location (IY + d).

Function:



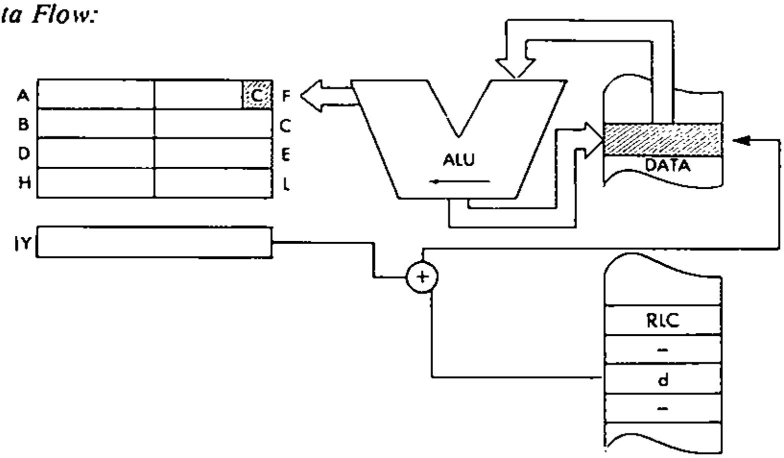
Format:

1	1	1	1	1	1	0	1	byte 1: FD
1	1	0	0	1	0	1	1	byte 2: CB
d								byte 3: offset value
0	0	0	0	0	1	1	0	byte 4: 06

Description:

The contents of the memory location addressed by the contents of the IY register plus the given offset value are rotated left and the result is stored back at the location. The contents of bit 7 are moved to the carry flag as well as bit 0.

Data Flow:



Timing: 6 M cycles; 23 T states; 11.5 usec @ 2 MHz

Addressing Mode: Indexed.

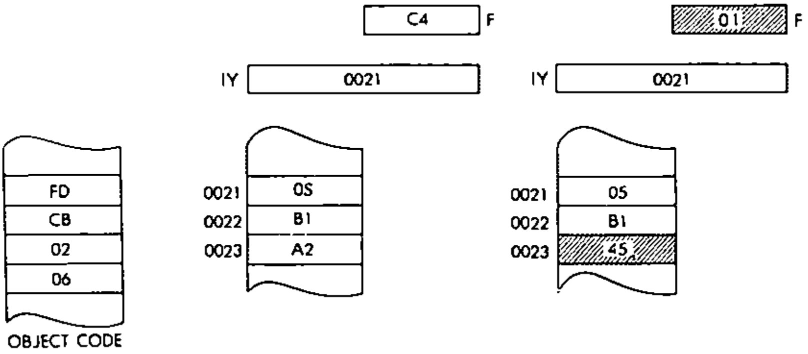
Flags:

S	Z		H		P/V	N	C
●	●		○		●	○	●

C is set by bit 7 of memory location.

Example: RLC (IY + 2)

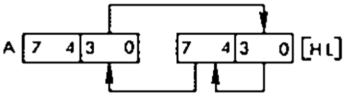
Before: After:



RLD

Rotate left decimal.

Function:

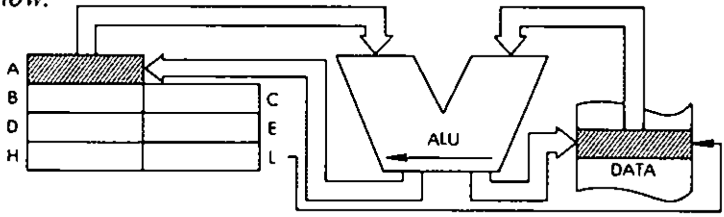


Format:

1	1	1	0	1	1	0	1	byte 1: ED
0	1	1	0	1	1	1	1	byte 2: 6F

Description: The 4 low order bits of the memory location addressed by the contents of HL are moved to the high order bit positions of that same location. The 4 high order bits are moved to the 4 low order bits of the accumulator. The low order of the accumulator is moved to the 4 low order bits of the memory location originally specified. All of these operations occur simultaneously.

Data Flow:



Timing: 5 M cycles; 18 T states; 9 usec @ 2 MHz

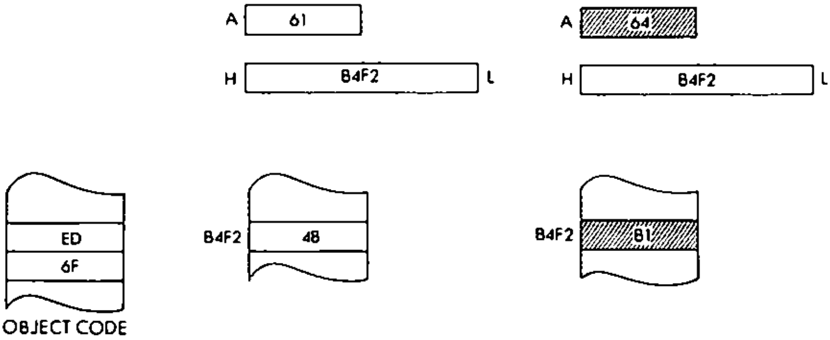
Addressing Mode: Indirect.

Flags:

S	Z		H		ⓅV	N	C
●	●		○		●	○	

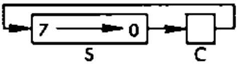
Examples: RLD

Before: After:



RR s Rotate right s through carry.

Function:



Format:

r	1	1	0	0	1	0	1	1	byte 1: CB
	0	0	0	1	1	←	r	→	byte 2
(HL)	1	1	0	0	1	0	1	1	byte 1: CB
	0	0	0	1	1	1	1	0	byte 2: 1E
(IX + d)	1	1	0	1	1	1	0	1	byte 1: DD
	1	1	0	0	1	0	1	1	byte 2: CB
	←			d				→	byte 3: offset value
	0	0	0	1	1	1	1	0	byte 4: 1E
(IY + d)	1	1	1	1	1	1	0	1	byte 1: FD
	1	1	0	0	1	0	1	1	byte 2: CB
	←			d				→	byte 3: offset value
	0	0	0	1	1	1	1	0	byte 4: 1E

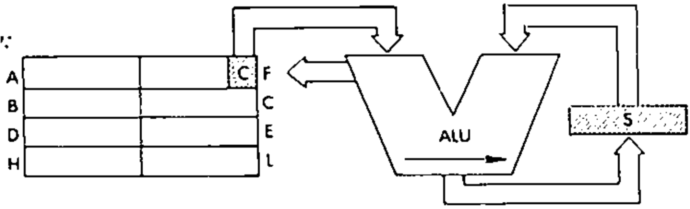
r may be any one of:

A – 111
B – 000
C – 001
D – 010

E – 011
H – 100
L – 101

Description: The contents of the location determined by the specific operand are shifted right. The contents of the carry flag are moved to bit 7 and the contents of bit 0 are moved to the carry flag. The final result is stored back in the original location. s is defined in the description of the similar RLC instructions.

Data Flow:



Timing:

s:	*M cycles:*	*T states:*	*usec @ 2 MHz:*
r	2	8	4
(HL)	4	15	7.5
(IX + d)	6	23	11.5
(IY + d)	6	23	11.5

Addressing Mode: r: implicit; (HL): indirect; (IX + d), (IY + d): indexed.

Byte Codes: RR r:

r:	A	B	C	D	E	H	L
CB-	1F	1B	19	1A	1B	1C	1D

Flags:

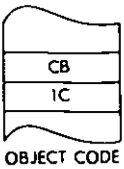
S	Z		H		P/V	N	C
●	●		○		●	○	●

C is set by bit 0 of source data.

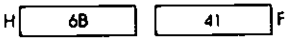
Example: RR H

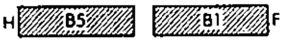
Before:

After:



OBJECT CODE

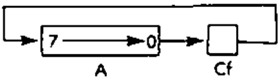




RRA

Rotate accumulator right through carry.

Function:

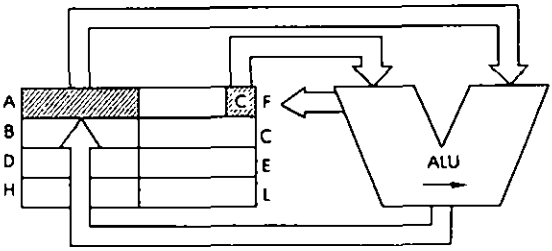


Format:

0	0	0	1	1	1	1	1	1F

Description: The contents of the accumulator are shifted right-one bit position. The contents of the carry flag are moved to bit 7 and the contents of bit 0 are moved to the carry flag (9-bit rotation).

Data Flow:



Timing: 1 M cycle; 4 T states; 2 usec @ MHz

Addressing Mode: Implicit.

Flags:

S	Z		H		P/V	N	C
			○			○	●

C is set by bit 0 of A.

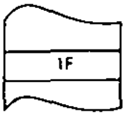
Example: RRA

Before:

A	F4	95	F

After:

A	FA	84	F



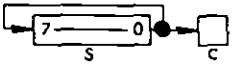
OBJECT CODE

Note: This instruction is almost identical to RR A. It is provided for 8080 compatibility.

RRC s

Rotate right with branch carry s.

Function:



Format: s: s is any of r, (HL), (IX + d), (IY + d).

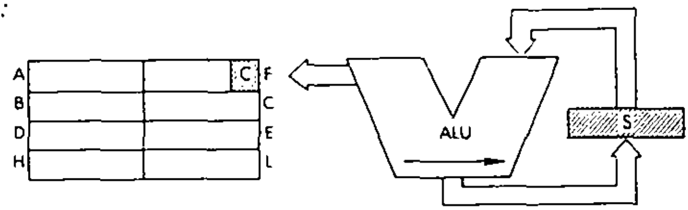
r	1	1	0	0	1	0	1	1	byte 1: CB
	0	0	0	0	1	← r →			byte 2
(HL)	1	1	0	0	1	0	1	1	byte 1: CB
	0	0	0	0	1	1	1	0	byte 2: 0E
(IX + d)	1	1	0	1	1	1	0	1	byte 1: DD
	1	1	0	0	1	0	1	1	byte 2: CB
	← d →								byte 3: offset value
	0	0	0	0	1	1	1	0	byte 4: 0E
(IY + d)	1	1	1	1	1	1	0	1	byte 1: FD
	1	1	0	0	1	0	1	1	byte 2: CB
	← d →								byte 3: offset value
	0	0	0	0	1	1	1	0	byte 4: 0E

r may be any one of:

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

Description: The contents of the location determined by the specified operand are rotated right and the result is stored back in the original location. The contents of bit 0 are moved to the carry flag as well as to bit 7. s is defined in the description of the similar RLC instructions.

Data Flow:



Timing:

s:	M cycles:	T states:	usec @ 2 MHz:
r	2	8	4
(HL)	4	15	7.5
(IX + d)	6	23	11.5
(IY + d)	6	23	11.5

Addressing Mode: *r:* implicit; (HL): indirect; (IX + d), (IY + d): indexed.

Byte codes: RRC r

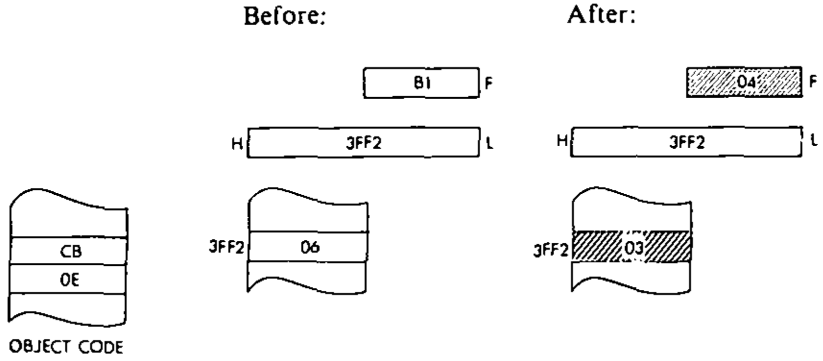
r:	A	B	C	D	E	H	L
CB-	0F	08	09	0A	0B	0C	0D

Flags:

S	Z		H		P/V	N	C
●	●		○		●	○	●

C is set by bit 0 of source data.

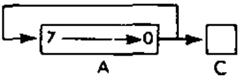
Example: RRC (HL)



RRCA

Rotate accumulator right with branch carry.

Function:

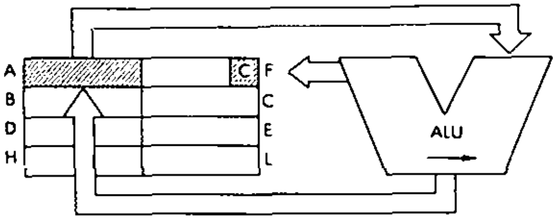


Format:

0	0	0	0	1	1	1	1	0F

Description: The contents of the accumulator are rotated right one bit position. The contents of bit 0 are moved to the carry flag as well as to bit 7.

Data Flow:



Timing: 1 M cycle; 4 T states; 2 usec @ 2 MHz

Addressing Mode: Implicit.

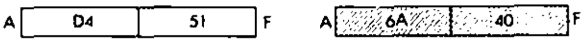
Flags:

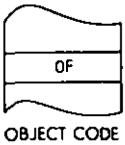
S	Z		H		P/V	N	C
			○			○	●

C is set by bit 0 of A.

Example: RRCA

Before: After:

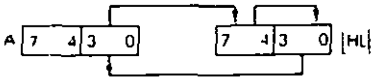




RRD

Rotate right decimal.

Function:

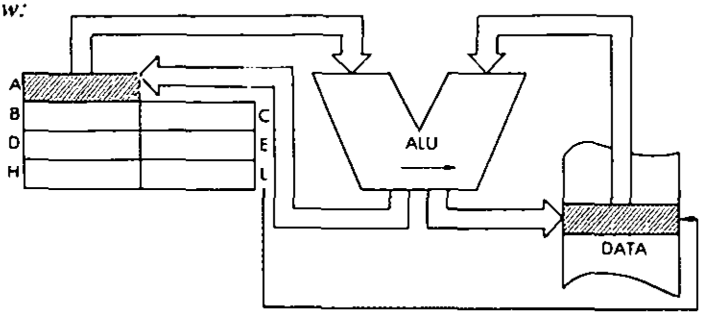


Format:

1	1	1	0	1	1	0	1	byte 1: ED
0	1	1	0	0	1	1	1	byte 2: 67

Description: The 4 high order bits of the memory location addressed by the contents of the HL register pair are moved to the low order 4 bits of that location. The 4 low order bits are moved to the 4 low order bits of the accumulator. The low order bits of the accumulator are moved to the 4 high order bit positions of the memory location originally specified. All of the above operations occur simultaneously.

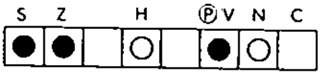
Data Flow:



Timing: 5 M cycles; 18 T states; 9 usec @ 2 MHz

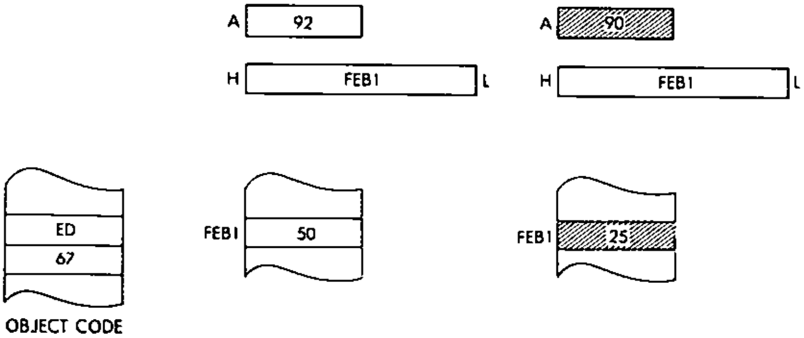
Addressing Mode: Indirect.

Flags:



Example: RRD

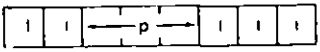
Before: After:



RST p Restart at p.

Function: $(SP - 1) \leftarrow PC_{high}$; $(SP - 2) \leftarrow PC_{low}$; $SP \leftarrow SP - 2$; $PC_{high} \leftarrow 0$; $PC_{low} \leftarrow p$

Format:

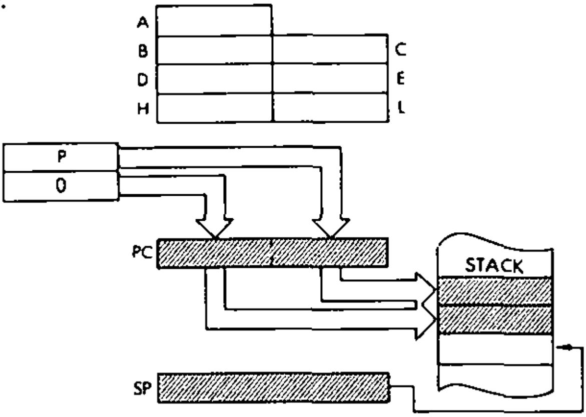


Description: The contents of the program counter are pushed onto the stack as described for the PUSH instructions. The specified value for p is then loaded into the PC and the next instruction is fetched from this new address. p may be any one of:

00H	000	20H	100
08H	001	28H	101
10H	010	30H	110
18H	011	38H	111

This instruction performs a jump to any of eight starting addresses in low memory and requires only a single byte. It may be used as a fast response to an interrupt.

Data Flow:



Timing: 3 M cycles; 11 T states; 5.5 usec @ 2 MHz

Addressing Mode: Indirect.

Byte Codes:

p:	00	08	10	18	20	28	30	38
	C7	CF	D7	DF	E7	EF	F7	FF

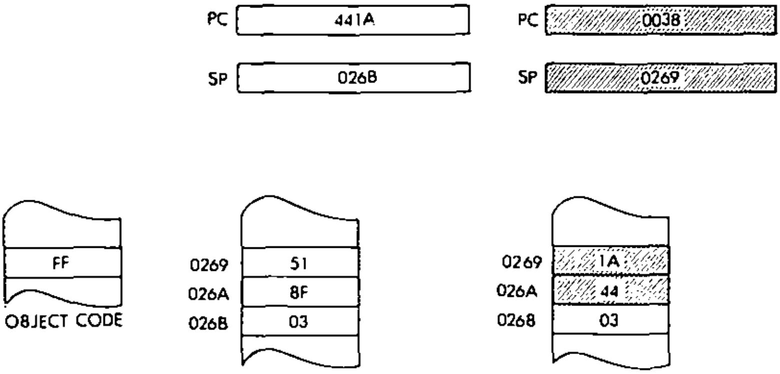
Flags:

S	Z		H		P/V	N	C

(no effect).

Example: RST 38H

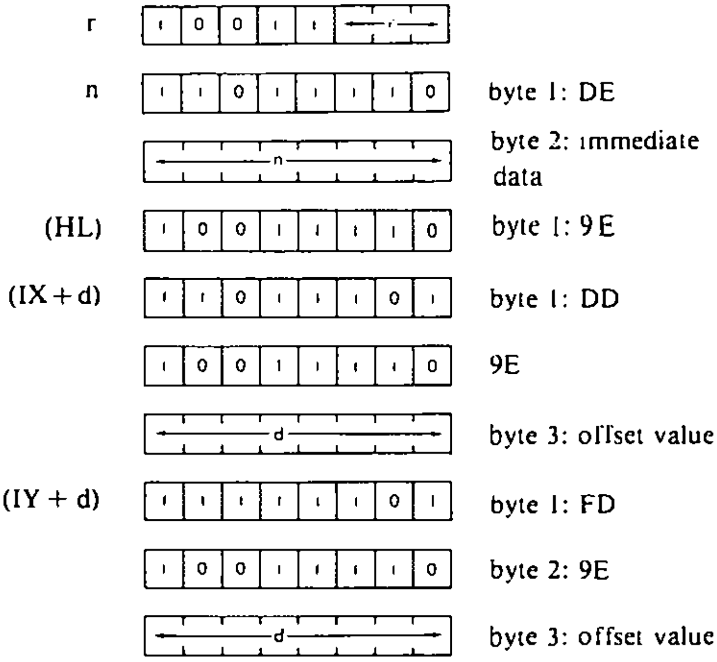
Before: After:



SBC A, s Subtract with borrow accumulator and specified operand.

Function: A ← A − s − C

Format: *s:* may be r, n, (HL), (IX + d), or (IY + d)

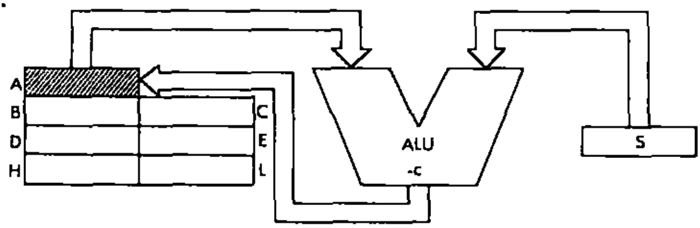


r may be any one of:

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

Description: The specified operand s, summed with the contents of the carry flag, is subtracted from the contents of the accumulator, and the result is placed in the accumulator. s is defined in the description of the similar ADD instructions.

Data Flow:



Timing:

s:	M cycles:	T states:	usec @ 2 MHz:
r	1	4	2
n	2	7	3.5
(HL)	2	7	3.5
(IX + d)	5	19	9.5
(IY + d)	5	19	9.5

Addressing Mode: r: implicit; n: immediate; (HL): indirect; (IX + d), (IY + d): indexed.

Byte Codes: SBC A, r

r: A	B	C	D	E	H	L
9F	98	99	9A	9B	9C	9D

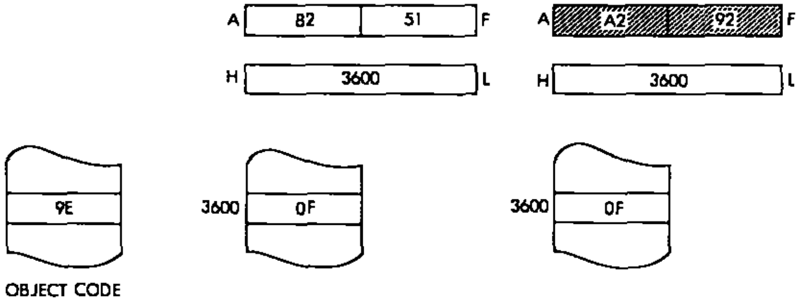
Flags:

S	Z		H		P/Ⓥ	N	C
●	●		●		●	1	●

Example: SBC A, (HL)

Before:

After:



SBC HL, ss

Subtract with borrow HL and register pair ss.

Function: HL ← HL − ss − C

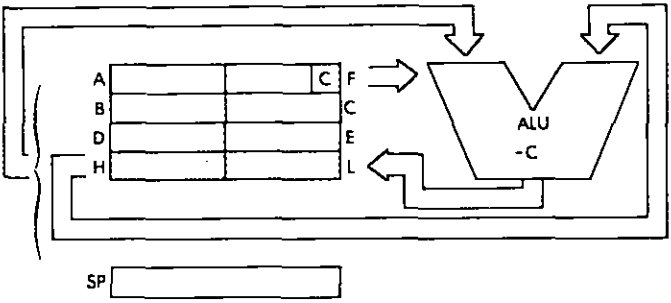
Format:

1	1	1	0	1	1	0	1	byte 1: ED
0	1	s	s	0	0	1	0	byte 2

Description: The contents of the specified register pair plus the contents of the carry flag are subtracted from the contents of the HL register pair and the result is stored back in HL. ss may be any one of:

BC − 00	HL − 10
DE − 01	SP − 11

Data Flow:



Timing: 4 M cycles; 15 T states; 7.5 usec @ 2 MHz

Addressing Mode: Implicit.

Byte Codes:

ss:	BC	OE	HL	SP
ED--	42	52	62	72

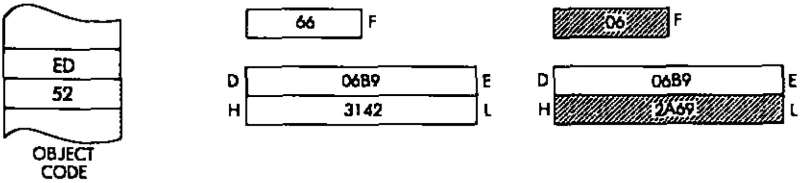
Flags:

S	Z		H		P/V	N	C
●	●		?		●	1	●

H is set if borrow from bit 12.
C is set if borrow.

Example: SBC HL, DE

Before: After:



SCF

Set carry flag.

Function: C ← 1

Format:

0	0	1	1	0	1	1	1	37

Description: The carry flag is set.

Timing: 1 M cycle; 4 T states; 2 usec @ 2 MHz

Addressing Mode: Implicit.

Flags:

S	Z		H		P/V	N	C
			0			0	1

SET b, s Set bit b of operand s

Function: $s_b \leftarrow 1$

Format: *s:*

r	1	1	0	0	1	0	1	1	byte 1: CB
	1	1	← b →			← r →			byte 2
(HL)	1	1	0	0	1	0	1	1	byte 1: CB
	1	1	← b →			1	1	0	byte 2
(IX + d)	1	1	0	1	1	1	0	1	byte 1: DD
	1	1	0	0	1	0	1	1	byte 2: CB
	← d →								byte 3: offset value
	1	1	← b →			1	1	0	byte 4
(IY + d)	1	1	1	1	1	1	0	1	byte 1: FD
	1	1	0	0	1	0	1	1	byte 2: CB
	← d →								byte 3: offset value
	1	1	← b →			1	1	0	byte 4

r may be any one of:

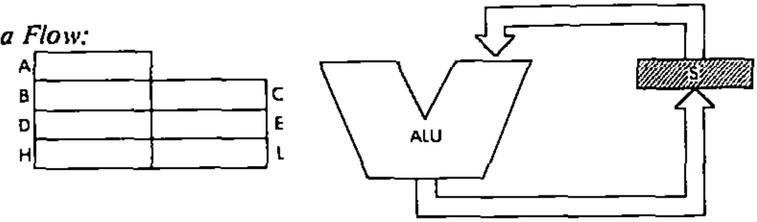
A – 111	E – 011
B – 000	H – 100
C – 001	L – 101
D – 010	

b may be any one of:

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

Description: The specified bit of the location determined by s is set. s is defined in the description of the similar BIT instructions.

Data Flow:



Timing:

s:	*M cycles:*	*T states:*	*usec @ 2 MHz:*
r	2	8	4
(HL)	4	15	7.5
(IX + d)	6	23	11.5
(IY + d)	6	23	11.5

Addressing Mode: r: implicit; (HL): indirect; (IX + d), (IY + d): indexed.

Byte Codes: SET b, r

	b: r:	A	B	C	D	E	H	L
CB–	0	C7	C0	C1	C2	C3	C4	C5
	1	CF	C8	C9	CA	CB	CC	CD
	2	D7	D0	D1	D2	D3	D4	D5
	3	DF	D8	D9	DA	DB	DC	DD
	4	E7	E0	E1	E2	E3	E4	E5
	5	EF	E8	E9	EA	EB	EC	ED
	6	F7	F0	F1	F2	F3	F4	F5
	7	FF	F8	F9	FA	FB	FC	FD

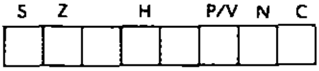
SET b, (HL)

SET b, (IX + d)

SET b, (IY + d)

b:	0	1	2	3	4	5	6	7
	C6	CE	D6	DE	E6	EE	F6	FE

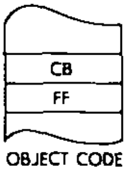
Flags:

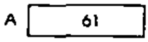


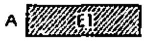
(no effect)

Example: SET 7, A

Before: After:



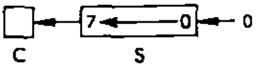




SLA s

Arithmetic shift left operand s.

Function:



Format: *s:*

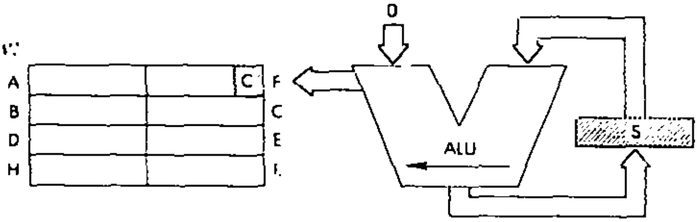
r	1	1	0	0	1	0	1	1	byte 1: CB
	0	0	1	0	0	←	r	→	byte 2
(HL)	1	1	0	0	1	0	1	1	byte 1: CB
	0	0	1	0	0	1	1	0	byte 2: 26
(IX + d)	1	1	0	1	1	1	0	1	byte 1: DD
	1	1	0	0	1	0	1	1	byte 2: CB
	←			d				→	byte 3: offset value
	0	0	1	0	0	1	1	0	byte 4: 26
(IY + d)	1	1	1	1	1	1	0	1	byte 1: FD
	1	1	0	0	1	0	1	1	byte 2: CB
	←			d				→	byte 3: offset value
	0	0	1	0	0	1	1	0	byte 4: 26

r may be any one of:

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

Description: The contents of the location determined by the specific operand are arithmetically shifted left with the contents of bit 7 being moved to the carry flag and a 0 being forced into bit 0. The final result is stored back in the original location. s is defined in the description of the similar RLC instructions.

Data Flow:



Timing:

s:	*M cycles:*	*T states:*	*usec @ 2 MHz:*
r	2	8	4
(HL)	4	15	7.5
(IX + d)	6	23	11.5
(IY + d)	6	23	11.5

Addressing Mode: r: implicit; (HL): indirect; (IX + d), (IY + d): indexed.

Byte Codes: SLA r

r:	A	B	C	D	E	H	L
CB-	27	20	21	22	23	24	25

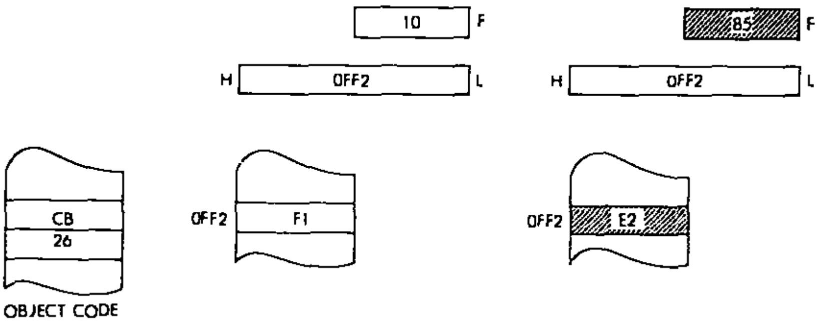
Flags:

S	Z		H		P/V	N	C
●	●		○		●	○	●

C is set by bit 7 of source data.

Example: SLA (HL)

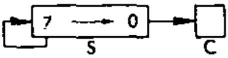
Before: After:



SRA s

Shift right arithmetic s.

Function:



Format:

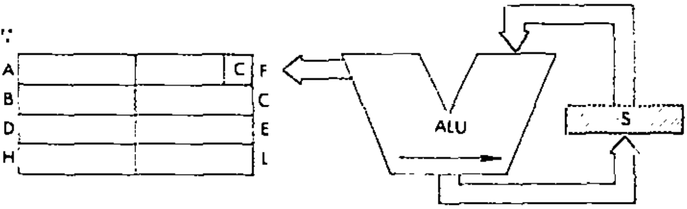
s:									
r	1	1	0	0	1	0	1	1	byte 1: CB
	0	0	1	0	1	← r →			byte 2
(HL)	1	1	0	0	1	0	1	1	byte 1: CB
	0	0	1	0	1	1	1	0	byte 2: 2E
(IX + d)	1	1	0	1	1	1	0	1	byte 1: DD
	1	1	0	0	1	0	1	1	byte 2: CB
	← d →								byte 3: offset value
	0	0	1	0	1	1	1	0	byte 4: 2E
(IY + d)	1	1	1	1	1	1	0	1	byte 1: FD
	1	1	0	0	1	0	1	1	byte 2: CB
	← d →								byte 3: offset value
	0	0	1	0	1	1	1	0	byte 4: 2E

r may be any one of:

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

Description: The contents of the location determined by the specific operand are arithmetically shifted right. The contents of bit 0 are moved to the carry flag and the contents of bit 7 remain unchanged. The final result is stored at the original location. s is defined in the description of the similar RLC instructions.

Data Flow:



Timing:

s:	*M cycles:*	*T states:*	*usec @ 2 MHz:*
r	2	8	4
(HL)	4	15	7.5
(IX + d)	6	23	11.5
(IY + d)	6	23	11.5

Addressing Mode: r: implicit; (HL): indirect; (IX + d), (IY + d): indexed.

Byte Codes: SRA r

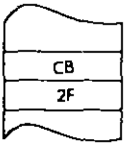
r:	A	B	C	D	E	H	L
CB-	2F	28	29	2A	2B	2C	2D

Flags:

S	Z		H		P/V	N	C
●	●		○		●	○	●

C is set by bit 0 of source data.

Example: SRA A



OBJECT CODE

Before:

A		F
8B	04	

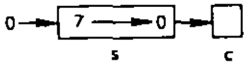
After:

A		F
C5	85	

SRL s

Logical shift right s.

Function:



Format:

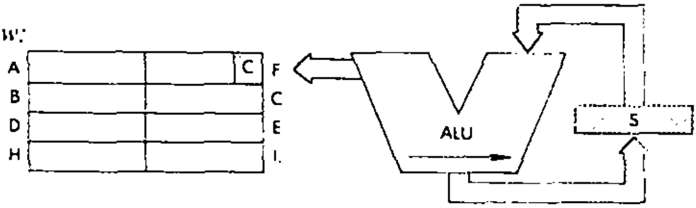
s:									
r	1	1	0	0	1	0	1	1	byte 1: CB
	0	0	1	1	1	←	r	→	byte 2
(HL)	1	1	0	0	1	0	1	1	byte 1: CB
	0	0	1	1	1	1	1	0	byte 2: 3E
(IX + d)	1	1	0	1	1	1	0	1	byte 1: DD
	1	1	0	0	1	0	1	1	byte 2: CB
	←			d				→	byte 3: offset value
	0	0	1	1	1	1	1	0	byte 4: 3E
(IY + d)	1	1	1	1	1	1	0	1	byte 1: FD
	1	1	0	0	1	0	1	1	byte 2: CB
	←			d				→	byte 3: offset value
	0	0	1	1	1	1	1	0	byte 4: 3E

r may be any one of:

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

Description: The contents of the location determined by the specific operand are logically shifted right. A zero is moved into bit 7 and the contents of bit 0 are moved into the carry flag. The final result is stored back in the original location.

Data Flow:



Timing:

s:	*M cycles:*	*T states:*	*usec @ 2 MHz:*
r	2	8	4
(HL)	4	15	7.5
(IX + d)	6	23	11.5
(IY + d)	6	23	11.5

Addressing Mode: r: implicit; (HL): indirect; (IX + d), (IY + d): indexed.

Byte Codes: SRL r

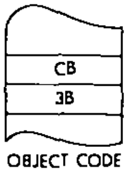
r:	A	B	C	D	E	H	L
CB	3F	38	39	3A	3B	3C	3D

Flags:

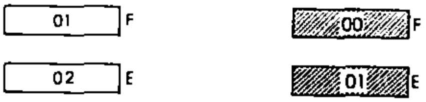
S	Z		H		P/V	N	C
●	●		○		●	○	●

C is set by bit 0 of source data.

Example: SRL E



Before: After:



SUB s Subtract operand s from accumulator.

Function: A ← A − s

Format: *s:* may be r, n, (HL), (IX + d) or (IY + d)

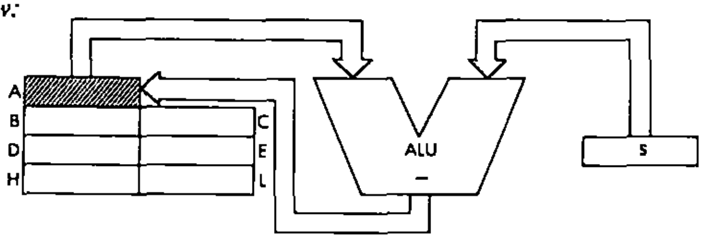
r	1	0	0	1	0	← r →			
n	1	1	0	1	0	1	1	0	byte 1: D6
	← n →								byte 2: immediate data
(HL)	1	0	0	1	0	1	1	0	96
(IX + d)	1	1	0	1	1	1	0	1	byte 1: DD
	1	0	0	1	0	1	1	0	byte 2: 96
	← d →								byte 3: offset value
(IY + d)	1	1	1	1	1	1	0	1	byte 1: FD
	1	0	0	1	0	1	1	0	byte 2: 96
	← d →								byte 3: offset value

r may be any one of:

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

Description: The specified operand s is subtracted from the accumulator and the result is stored in the accumulator. The operand s is defined in the description of the similar ADD instructions.

Data Flow:



Timing:

s:	*M cycles:*	*T states:*	*usec @ 2 MHz:*
r	1	4	2
n	2	7	3.5
(HL)	2	7	3.5
(IX + d)	5	19	9.5
(IX + d)	5	19	9.5

Addressing Mode: r: implicit; n: immediate; (HL): indirect; (IX + d), (IY + d): indexed

Byte Codes: SUB r

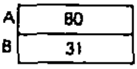
r:	A	B	C	D	E	H	L
	97	90	91	92	93	94	95

Flags:

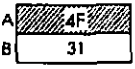
S	Z		H		P/Ⓥ	N	C
●	●		●		●	1	●

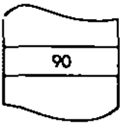
Example: SUB B

Before:



After:





OBJECT CODE

XOR s

Exclusive or accumulator and s.

Function: $A \leftarrow A \veebar s$

Format: s: may be r, n, (HL), (IX + d), or (IY + d)

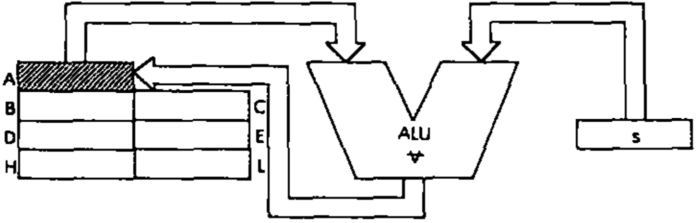
r	1 0 1 0 1 ← r →	
n	1 1 1 0 1 1 1 0	byte 1: EE
	← n →	byte 2: immediate data
(HL)	1 0 1 0 1 1 1 0	AE
(IX + d)	1 1 0 1 1 1 0 1	byte 1: DD
	1 0 1 0 1 1 1 0	byte 2: AE
	← d →	byte 3: offset value
(IY + d)	1 1 1 1 1 1 0 1	byte 1: FD
	1 0 1 0 1 1 1 0	byte 2: AE
	← d →	byte 3: offset value

r may be any one of:

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

Description: The accumulator and the specified operand s are exclusive 'or'ed, and the result is stored in the accumulator. s is defined in the description of the similar ADD instructions.

Date Flow:



Timing:

s:	*M cycles:*	*T states:*	*usec @ 2 MHz:*
r	1	4	2
n	2	7	3.5
(HL)	2	7	3.5
(IX + d)	5	19	9.5
(IY + d)	5	19	9.5

Addressing Modes: r: implicit; n: immediate; (HL): indirect; (IX + d), (IY + d): indexed

Byte Codes: XOR r

r:

A	B	C	D	E	H	L
AF	A8	A9	AA	AB	AC	AD

Flags:

S	Z		H		P/V	N	C
●	●		○		●	○	○

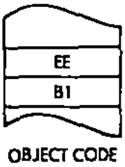
Example: XOR B1H

Before:

A 36

After:

A 87



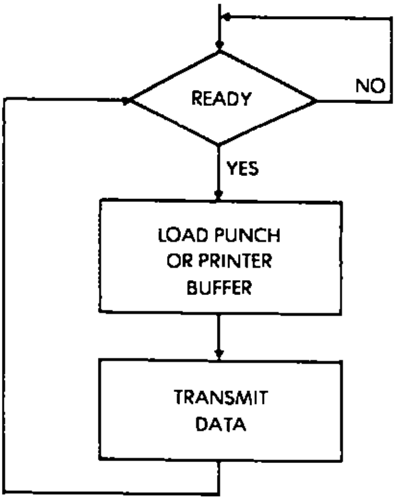


Fig. 6.21: Printing on a Punch or Printer

ter to do, as it keeps the overall organization simple. Let us examine the essential alternative to polling: interrupts.

Interrupts

The concept of interrupts is illustrated in Figure 6.18. A special hardware line, the interrupt line, is connected to a specialized pin of the microprocessor. Multiple input/output devices may be connected to this interrupt line. When any one of them needs service, it sends a level or a pulse on this line. An interrupt signal is the service request from an input/output device to the processor. Let us examine the response of the processor to this interrupt.

In any case, the processor completes the instruction that it was currently executing; otherwise, this would create chaos inside the microprocessor. Next, the microprocessor should branch to an interrupt-handling routine which will process the interrupt. Branching to such a subroutine implies that the contents of the program counter must be saved on the stack. *An interrupt must, therefore, cause the automatic preservation of the program counter on the stack.* In addition, the flag register F should be also preserved automatically, as its contents will be altered by any subsequent instruction. Finally, if the interrupt-handling routine should modify any internal registers, these internal registers should also be preserved on the stack (see Figures 6.22 and 6.23).

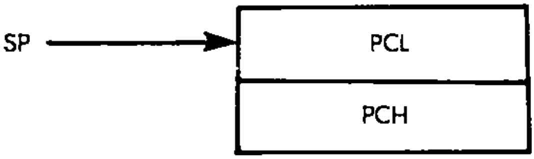


Fig. 6.22: Z80 Stack After Interruption

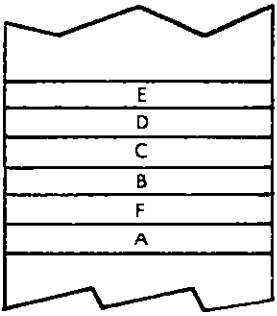


Fig. 6.23: Saving Some Working Registers

After all these registers have been preserved, one can branch to the appropriate interrupt-handling address. At the end of this routine, all the registers should be restored, and a special interrupt return should be executed so that the main program will resume execution. Let us examine in more detail the interrupt lines of the Z80.

Z80 Interrupts

An interrupt is a signal sent to the microprocessor, which may request service at any time and is asynchronous to the program. Whenever a program branches to a subroutine, such branching is *synchronous* to program execution, i.e., scheduled by the program. An interrupt, however, may occur at any time, and will generally suspend the execution of the current program (without the program knowing it). Because it may happen at any time relative to program execution, it is called *asynchronous*.

Three interruption mechanisms are provided on the Z80: the bus request (BUSRQ), the non-maskable interrupt (NMI) and the usual interrupt (INT).

Let us examine these three types.

The Bus Request

The bus request is the highest priority interrupt mechanism on the Z80. The interrupt sequence for the Z80 is shown in Figure 6.24. As a general rule, no interrupt will be sensed by the Z80 until the current machine cycle is completed. The NMI and INT interrupts will not be taken into account until the current instruction is finished. However, the BUSRQ will be handled at the end of the current machine cycle, without necessarily waiting for the end of the instruction. It is used for

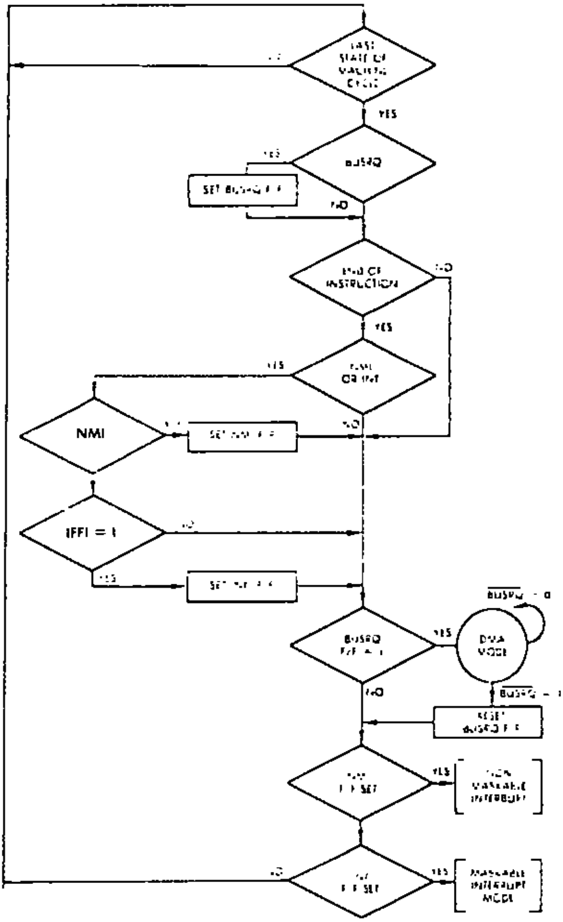


Fig. 6.24: Interrupt Sequence

a direct memory access (DMA), and will cause the Z80 to go into DMA mode (see ref. C201 for an explanation of the DMA mechanism). If the end of an instruction has been reached, and if any NMI or INT were pending, they would be memorized internally in the Z80 by setting specialized flip-flops: the NMI flip-flop, and the INT flip-flop. In DMA mode, the Z80 suspends operation and releases its data-bus and address-bus in the high-impedance state. This mode is normally used by a DMA controller to perform transfers between a high-speed input-output device and the memory, using the microprocessor data-bus and address-bus. The end of a DMA operation is indicated to the Z80 by BUSRQ changing levels. At this point, the Z80 will resume normal operation. In particular, it will first check whether its internal NMI or INT flip-flops had been set and, if so, execute the corresponding interrupts.

The DMA should normally not be of concern to the programmer, unless timing is important. If a DMA controller is present in the system, the programmer must understand that the DMA may delay the response to an NMI or an INT.

The Non-Maskable Interrupt

This type of interrupt cannot be inhibited by the programmer. It is therefore said to be *non-maskable*, hence its name. It will always be accepted by the Z80 upon completion of the current instruction, assuming no bus request was received. (If an NMI is received during a BUSRQ, it will set the internal NMI flip-flop, and will be processed at the end of the instruction following the end of the BUSRQ.)

The NMI will cause an automatic push of the program counter into the stack and branch to address 0066H: the two bytes representing the address 0066H will be installed in the program counter. They represent the start address of the handling routine for the NMI (see figure 6.25).

This interrupt mechanism has been designed for speed, as it is used in case of "emergencies". Therefore, it does not offer the flexibility of the maskable interrupt mode, described below.

Note also that an interrupt routine must have been loaded at address 0066H prior to using the NMI.

NMI will first cause:

$$\left.\begin{array}{l} \text{SP} \longleftarrow \text{SP} - 1 \\ \text{(SP)} \longleftarrow \text{PCH} \\ \text{SP} \longleftarrow \text{SP} - 1 \\ \text{(SP)} \longleftarrow \text{PCL} \end{array}\right\} \text{push PC}$$

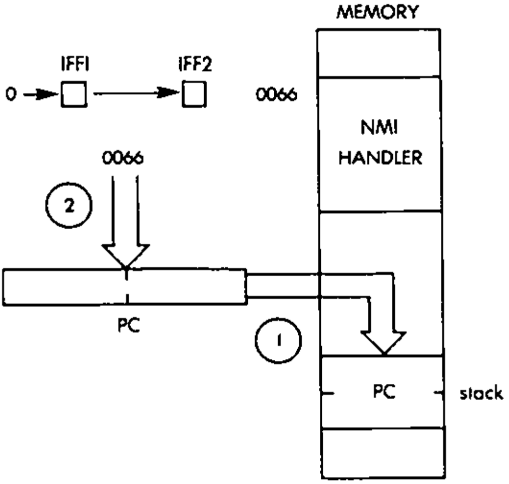


Fig. 6.25: NMI Forces Automatic Vectoring

Then, NMI causes an automatic restart at location 0066H. The complete sequence of events is the following:

PC	⟶	STACK	(preserve program counter)
IFFI	⟶	IFF2	(preserve IFF)
0	⟶	IFF1	(reset IFF)
JUMP TO 0066H			(execute interrupt handler)

Also, the status of interrupt-mask-bit flip-flop (IFF1) at the time that NMI was received is preserved automatically into IFF2. Then, IFFI is reset in order to prevent any further interrupts. This feature is important to prevent the loss of lower-priority INT's and simplifies the external hardware: the status of a pending INT is preserved internally in the Z80.

The NMI interrupt is normally used for high priority events such as a real-time clock or a power failure.

The return from an NMI is accomplished by a special instruction, RETN: "return from non-maskable interrupt." The contents of IFFI are restored from IFF2, and the contents of the program counter PC are restored from their location in the stack. Since IFF1 had been reset during execution of the NMI, no external INT's could be accepted during the NMI (unless the programmer uses an EI instruction within the NMI routine): there has been no loss of information.

Upon termination of the interrupt handler, the sequence is:

IFF2	⟶	IFFI	(restore IFF)
STACK	⟶	PC	(restore program counter)

Note that, once IFFI is restored, maskable interrupt enable status is restored.

Interrupt

The ordinary, maskable,,interrupt INT may operate in one of three modes. They are specific to the Z80, as the 8080 is equipped with only a single interrupt mode. The ordinary interrupt INT may also be masked selectively by the programmer. Setting the interrupt flip-flops IFF1 and IFF2 to a "1" will authorize interruptions. Setting them to a "0" (masking them) will prevent detection of INT. The EI instruction is used to set them, and the DI instruction is used to reset them. IFF1 and IFF2 are set or reset simultaneously. During execution of the EI and DI instructions, INT's are disabled in order to prevent any loss of information.

Let us now examine the three interrupt modes:

Interrupt Mode 0

This mode is identical to the 8080 interrupt mode. The Z80 will operate in interrupt mode 0 either when initially started (when the RESET signal has been applied) or else when an IM0 instruction has been executed. Once mode 0 has been set, an interrupt will be recognized if the interrupt enable flip-flop IFF1 is set to 1, provided no bus-request or non-maskable interrupt occurs at the same time. The interrupt will be detected only at the end of an instruction. Essentially, the Z80 will respond to the interrupt by generating an IORQ (and an M1 signal), and then do nothing, except wait.

It is the responsibility of an *external device* to recognize the IORQ and M1 (this is called an *interrupt acknowledge* or INTA) and to place an instruction on the data-bus. The Z80 expects an instruction to be placed on its data bus by the external device within the next cycle. Typically, an RST or a CALL instruction is placed on the bus. Both of these instructions automatically preserve the program-counter in the stack, and cause branching to a specific address. The advantage of the RST instruction is that it resides within a single byte, i.e., it executes rapidly. Its disadvantage is to branch to only one of eight possible locations in page zero (addresses 0 through 255). The advantage of the CALL instruction is that it is a general-purpose branch instruction which specifies a full 16-bit address. However, it requires three bytes and therefore executes less rapidly.

Note that once the interrupt processing starts, all further interrupts are disabled. IFF1 and IFF2 are automatically set to "0". It is then the responsibility of the programmer to insert an EI instruction (Enable Interrupts) at the appropriate location within his program if he wishes to enable interrupts, and, in any case, before returning from the interrupt.

The detailed sequence corresponding to the mode 0 interrupt is shown in Figure 6.26.

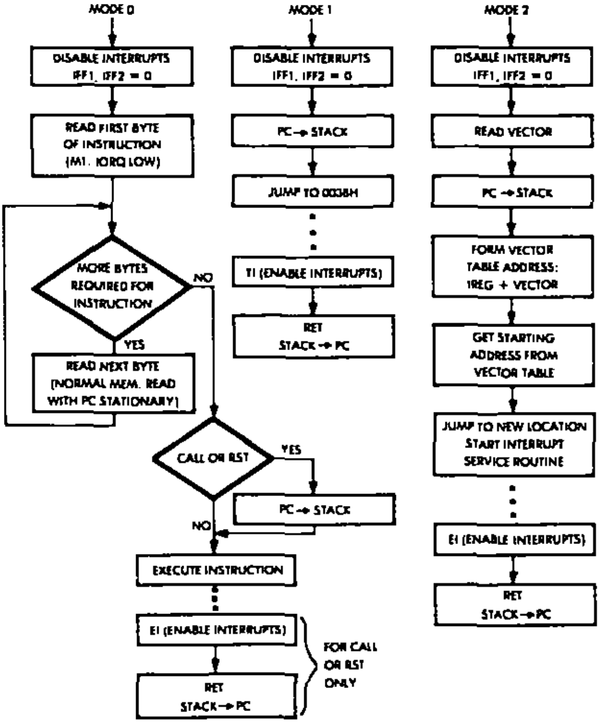

MODE 0
DISABLE INTERRUPTS IFF1, IFF2 = 0
READ FIRST BYTE OF INSTRUCTION (M1, IORQ LOW)
MORE BYTES REQUIRED FOR INSTRUCTION
NO
YES
READ NEXT BYTE (NORMAL MEM. READ WITH PC STATIONARY)
CALL OR RST
YES
PC → STACK
NO
EXECUTE INSTRUCTION
EI (ENABLE INTERRUPTS)
RET STACK → PC
FOR CALL OR RST ONLY
MODE 1
DISABLE INTERRUPTS IFF1, IFF2 = 0
PC → STACK
JUMP TO 0038H
TI (ENABLE INTERRUPTS)
RET STACK → PC
MODE 2
DISABLE INTERRUPTS IFF1, IFF2 = 0
READ VECTOR
PC → STACK
FORM VECTOR TABLE ADDRESS: IREG + VECTOR
GET STARTING ADDRESS FROM VECTOR TABLE
JUMP TO NEW LOCATION START INTERRUPT SERVICE ROUTINE
EI (ENABLE INTERRUPTS)
RET STACK → PC

Fig. 6.26: Interrupt Modes

The return from the interrupt is accomplished by an RETI instruction. Let us remind the programmer at this point that he/she is usually responsible for explicitly clearing the interrupt which has been serviced on the I/O device, and always for restoring the interrupt disable flag inside the Z80. However, the peripheral controller may use the INTA signal to clear the INT request, thus freeing the programmer of this chore.

In addition, should the interrupt-handling routine modify the contents of any of the internal registers, the programmer is specifically responsible for preserving these registers in the stack prior to executing the interrupt-handling routine. Otherwise, the contents of these registers will be destroyed, and when the interrupted program resumes execution, it will fail. For example, assuming that registers A, B, C, D, E, H and L will be used within the interrupt handler, they will have to be saved (see Figure 6.27).

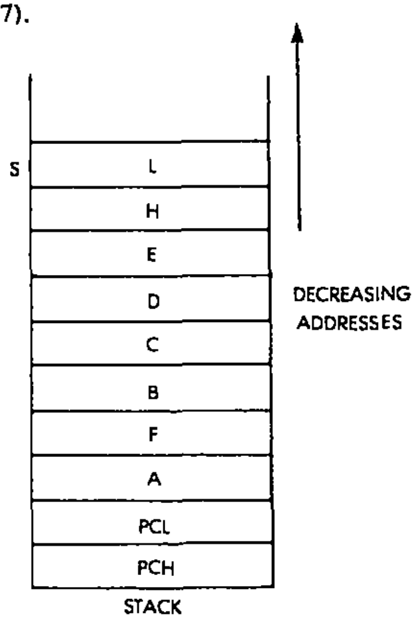


Fig. 6.27: Saving the Registers

The corresponding program is:

```
SAVREG   PUSH  AF
         PUSH  BC
         PUSH  DE
         PUSH  HL
```

Upon completion of the interrupt-handling routine, these registers must be restored. The interrupt handler will terminate with the following sequence of instructions:

```
         POP   HL
         POP   DE
         POP   BC
         POP   AF
         EI                   (unless EI was used earlier in
                              the routine)
```

Additionally, if registers IX and IY are used by the routine they must also be preserved, then restored.

Interrupt Mode 1

This interrupt mode is set by executing the IM1 instruction. It is an automated interrupt handler which causes an automatic branch to location 0038H. It is therefore essentially analogous to the NMI interrupt mechanism except that it may be masked. The Z80 automatically preserves the contents of PC into the stack (see Figure 6.28).

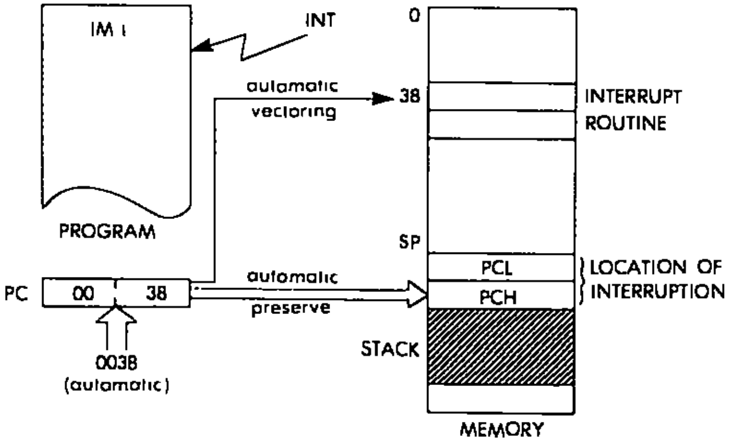


Fig. 6.28: Mode 1 Interrupt

This automated interrupt response, which "vectors" all interrupts to memory location 38H, stems from the early 8080's requirement to minimize the amount of external hardward necessary for using interrupts. Its possible disadvantage is to cause a branch to a *single* memory location. In case several devices are connected to the INT line, the program starting at location 38H will be responsible for determining which device requested service. This problem will be addressed below.

One precaution must be taken with respect to the timing of this interrupt: when performing programmed input/output transfers, the Z80 will ignore any data that may be present in the data bus during the cycle which follows the interrupt (the interrupt acknowledge cycle).

Interrupt Mode 2 (Vectored Interrupts)

This mode is set by executing an IM2 instruction. It is a powerful mode which allows automatic vectoring of interrupts. The interrupt vector is an address supplied by the peripheral device which generated the interrupt, and used as a memory pointer to the start address of the interrupt-handling routine. The addresssing mechanism provided by the Z80 in mode 2 is indirect, rather than direct. Each peripheral supplies a seven-bit branching address which is appended to the 8-bit address contained in the special I register in the Z80. The right-most bit of the final 16-bit address bit 0 is set to "0". This resulting address points to an entry in a table anywhere in the memory. This table may contain up to 128 double-word entries. Each of these double words is the address of the interrupt handler for the corresponding device. This is illustrated in Figures 6.29 and 6.30.

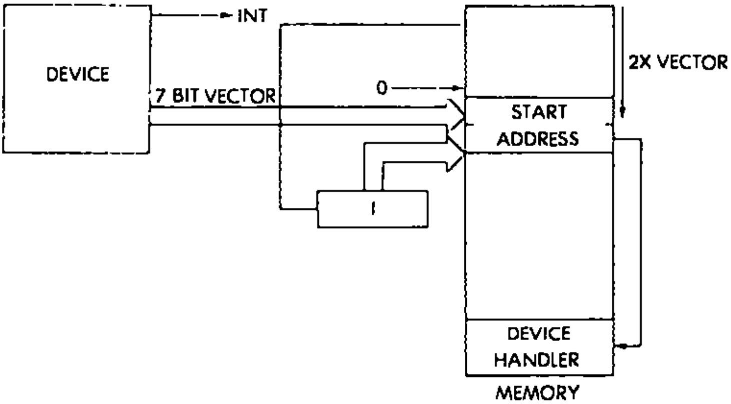


Fig. 6.29: Mode 2 Interrupt

The interrupt table may have up to 128 double-word entries.

In this mode, the Z80 also automatically pushes the contents of the program counter into the stack. This is obviously necessary, since PC will be reloaded with the contents of the interrupt table entry corresponding to the vector provided by the device.

Interrupt Overhead

For a graphic comparison of the polling process vs. the interrupt process, refer to Figure 6.18, where the polling process is illustrated on the top, and the interrupt process underneath. It can be seen that in the polling technique the program wastes a lot of time waiting.

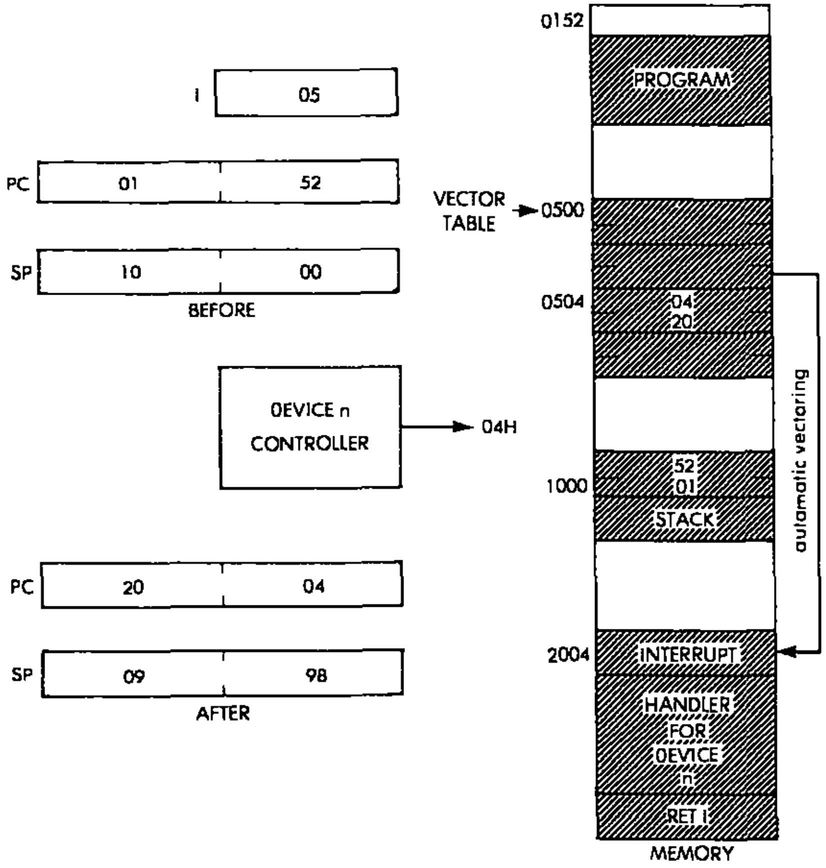


Fig. 6.30: Mode 2.- A Practical Example

Using interrupts, the program is interrupted, the interrupt is serviced, then the program resumes. However, the obvious disadvantage of an interrupt is to introduce several additional instructions at the beginning and at the end, resulting in a delay before the first instruction of the device handler can be executed. This is additional overhead.

***Exercise 6.28:** Using the tables indicating the number of cycles per instruction, in Chapter 4, compute how much time will be lost to save and then restore registers A, B, D, H.*

Having clarified the operation of the interrupt lines, let us now consider two important remaining problems:

1—How do we resolve the problem of multiple devices triggering an interrupt at the same time?

2—How do we resolve the problem of an interrupt occurring while another interrupt is being serviced?

Multiple Devices Connected to a Single Interrupt Line

Whenever an interrupt occurs, the processor branches to a specified address. Before it can do any effective processing, the interrupt handling routine must determine which device triggered the interrupt. Two methods are available to identify the device, as usual: a software method and a hardware method.

In the software method, polling is used: the microprocessor interrogates each of the devices in turn and asks them, "Did you trigger the interrupt?" If the answer is negative, it interrogates the next one. This process is illustrated in Figure 6.31. A sample program is:

```
POLINT  IN    A, (STATUS1)  READ STATUS
        BIT   7, A          DID DEVICE REQUEST INT?
        JP    NZ, ONE       HANDLE IT IF SO
        IN    A, (STATUS2)
        BIT   7, A
        JP    NZ, TWO
        etc.  ---
```

The hardward method provides the address of the interrupting device simultaneously with the interrupt request.

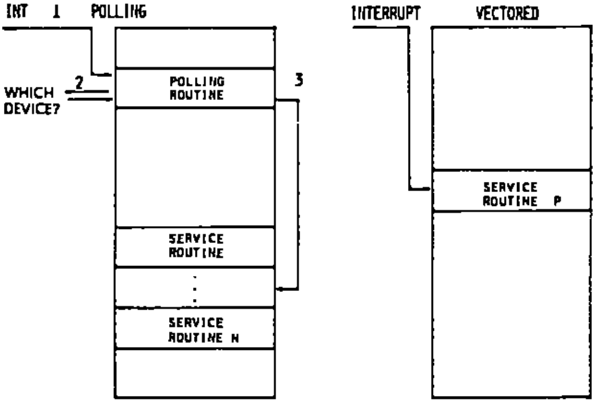


Fig. 6.31: Polled vs. Vectored Interrupt

To be more precise, when operating in mode 0, the peripheral device controller will supply a one-byte RST or a three-byte CALL on the data bus in response to the interrupt acknowledge, thus automating the interrupt vectoring, and minimizing the overhead.

Note that a subroutine call instruction is required as the Z80 does not save the PC when operating in mode 0.

In most cases, the speed of reaction to an interrupt is not crucial, and a polling approach is used. If response time is a primary consideration, a hardware approach must be used.

Simultaneous Interrupts

The next problem which may occur is that a new interrupt can be triggered during the execution of an interrupt-handling routine. Let us examine what happens and how the stack is used to solve the problem. We have indicated in Chapter 2 that this was another essential role of the stack, and the time has come now to demonstrate its use. We will refer to Figure 6.33 to illustrate multiple interrupts. Time elapses from left to right in the illustration. The contents of the stack are shown at the bottom of the illustration. Looking at the left, at time T0, program P is in execution. Moving to the right, at time T1, interrupt I1 occurs. We will assume that the interrupt mask was enabled, authorizing I1. Program P will be suspended. This is shown at the bottom of the illustration. The stack will contain the program counter and the status register of program P, at least, plus any optional registers that might be saved by the interrupt handler or I1 itself.

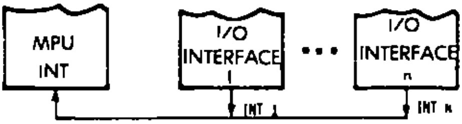


Fig. 6.32: Several Devices May Use the Same Interrupt Line

At time T1, interrupt I1 starts executing until time T2. At time T2, interrupt I2 occurs. We will assume that interrupt I2 has a higher priority than interrupt I1. If it had a lower priority, it would be ignored until I1 had been completed. At time T2, the registers for I1 are stacked, and this appears at the bottom of the illustration. Again, the contents of the program counter and AF are pushed into the stack. In addition, the routine for I2 might decide to save an additional few registers. I2 will now execute to completion at time T3.

When 12 terminates (with an RETI), the contents of the stack are automatically popped back into the Z80, and this is illustrated at the bottom of Figure 6.33. Thus, automatically I1 resumes execution. Unfortunately, at time T4, an interrupt 13 of higher priority occurs again. We can see at the bottom of the illustration that again the registers for I1 are pushed into the stack. Interrupt 13 executes from T4 to T5 and terminates at T5. At that time, the contents of the stack are popped into Z80, and interrupt I1 resumes execution. This time it runs to completion and terminates at T6. At T6, the remaining registers that have been saved in the stack are popped into Z80, and progam P may resume execution. The reader will verify that the stack is empty at this point. In fact, the number of dashed lines indicating program suspension indicates at the same time how many levels there are in the stack.

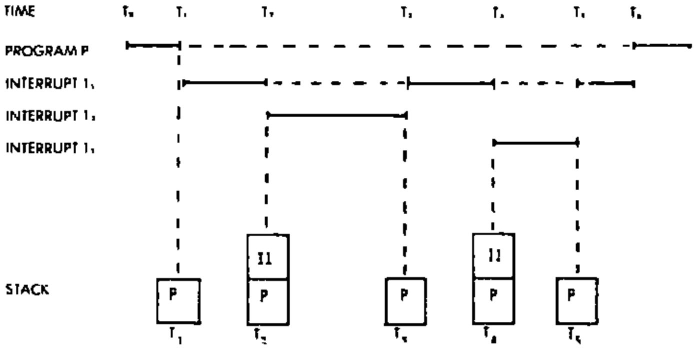


Fig. 6.33: Stack Contents During Multiple Interrupts

Exercise 6.29: Assume that the area available to the stack is limited to 300 locations in a specific program. Assume that all the registers must always be saved and that the programmer allows interrupts to be nested, i.e., to interrupt each other. Which is the maximum number of simultaneous interrupts that can be handled? Will any other factor contribute to still reduce further the maximum number of simultaneous interrupts?

It must be stressed, however, that, in practice, microprocessor systems are normally connected to a small number of devices using interrupts. It is, therefore, unlikely that a high number of simultaneous interrupts will occur in such a system.

We have now solved all the problems usually associated with interrupts. Their use is, in fact, simple and they should be employed to advantage even by the novice programmer.