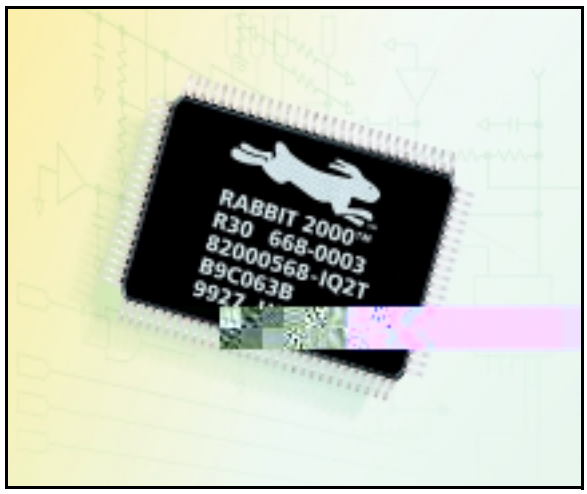


Rabbit 2000™ Microprocessor

Instruction Reference Manual

019–0098 A • 010330

Rabbit 2000 Microprocessor Instruction Reference Manual

Part Number 019–0098 A • 010330 • Printed in U.S.A.

Notice to Users

Rabbit Semiconductor products are not authorized for use as critical components in life-support devices or systems unless a specific written agreement regarding such intended use is entered into between the customer and Rabbit Semiconductor prior to use. Life-support devices or systems are devices or systems intended for surgical implantation into the body or to sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling and user's manual, can be reasonably expected to result in significant injury.

No complex software or hardware system is perfect. Bugs are always present in a system of any size. In order to prevent danger to life or property, it is there responsibility of the system designer to incorporate redundant protective mechanisms appropriate to the risk involved.

Rabbit Semiconductor
2932 Spafford Street
Davis, California 95616-6800
USA

Telephone (530) 757-8400
Fax (530) 757-8402

http://www.rabbitsemiconductor.com

Table of Contents

1. Instruction Listing by Groups

A. Load Immediate Data

B. Load and Store to an Immediate Address

C. 8-bit Indexed Load and Store

D. 16-bit Indexed Load and Store

E. 16-bit Load and Store to 20-bit Address

F. Register to Register Moves

G. Exchange

H. Stack Manipulation

I. 16-bit Arithmetic, Logical, and Rotate

INDEX

J. 8-bit Arithmetic and Logical

K. 8-bit Bit Set, Reset, and Test

L. 8-bit Increment and Decrement

M. 8-bit Fast Accumulator

N. 8-bit Shift and Rotate

O. Instruction Prefixes

P. Block Moves

Q. Control, Jump, and Call

R. Miscellaneous

S. New Instructions

T. Privileged Instructions

2. Document Conventions

Function Heading Key

In the tinted heading for each function (such as the example shown below), letter flags at the far right indicate that the function is new (implemented in the Rabbit 2000, but not previously available in the Z180 instruction set), modified from its Z180 equivalent, or privileged. (Some instructions are both new and privileged.)

No flags in this location indicate that the instruction is identical or very similar to its Z180 counterpart, and not a privileged instruction.

RETI NP

M = Modified

N = New

P = Privileged

NP = New & Privileged

Instruction Table Key

- **Opcode**: A hexidecimal representation of the value that the mnemonic instruction represents.
- **Instruction**: The mnemonic syntax of the instruction.
- **Clocks**: The number of clock cycles it takes to complete this instruction. The numbers in parenthesis are a breakdown of the total clocks. The number of clocks instructions take follows a general patern. There are several Rabbit instructions that do not adhere to this pattern. Some instructions take more clocks and some have been enhanced to take fewer clocks.

Table 1: Clocks Breakdownm

- **Operation**: A symbolic representation of the operation performed.

ALTD, I/O and Flags Table Keys

Table 2: ALTD (“A” Column) Symbol Key

Flag			Description
F	R	SP	
•			ALTD selects alternate flags
	•		ALTD selects alternate destination register
		•	ALTD operation is a special case

Table 3: IOI and IOE (“I” Column) Symbol Key

Flag		Description
S	D	
	•	IOI and IOE affect destination
•		IOI and IOE affect source

Table 4: Flag Register Key

S	Z	L/V	C	Description
•				Sign flag affected
-				Sign flag not affected
	•			Zero flag affected
	-			Zero flag not affected
		L		LV flag contains logical check result
		V		LV flag set on arithmetic overflow result
		0		LV flag is cleared
		•		LV flag is affected
			•	Carry flag is affected
			-	Carry flag is not affected
			0	Carry flag is cleared
			1	Carry flag is set

Document Symbols Key

Table 5: Symbols

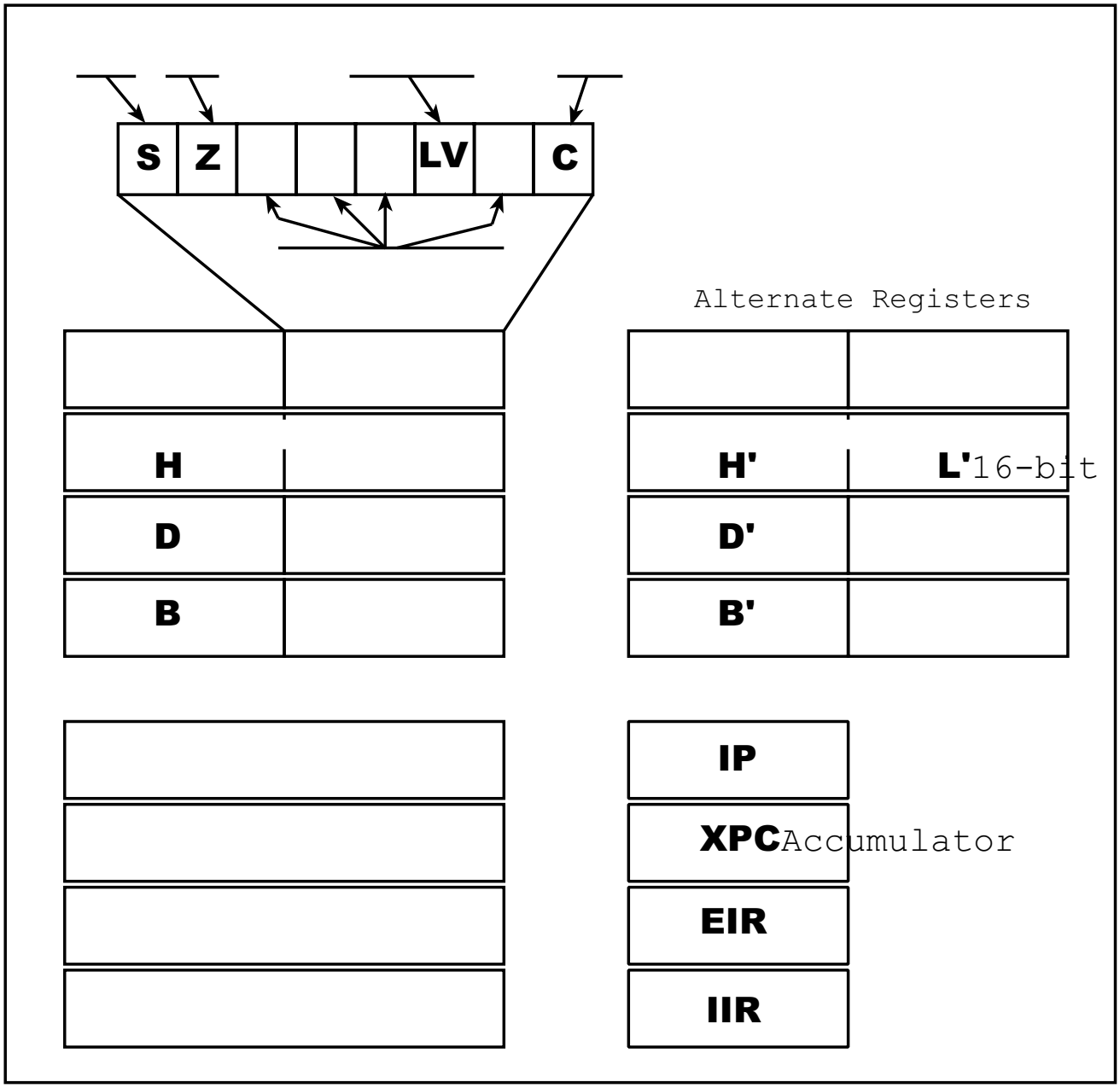
Rabbit	Z180	Meaning
b	*b*	Bit select (000 = bit 0, 001 = bit 1, 010 = bit 2, 011 = bit 3, 100 = bit 4, 101 = bit 5, 110 = bit 6, 111 = bit 7)
cc	*cc*	Condition code select (00 = NZ, 01 = Z, 10 = NC, 11 = C)
d	*d*	7-bit (signed) displacement. Expressed in two's complement.
dd	*ww*	word register select-destination (00 = BC, 01 = DE, 10 = HL, 11 = SP)
dd'		word register select-alternate(00 = BC', 01 = DE', 10 = HL')
e	*j*	8-bit (signed) displacement added to PC
f	*f*	condition code select (000 = NZ, 001 = Z, 010 = NC, 011 = C, 100 = LZ/NV, 101 = LO/V, 110 = P, 111 = M)
m	*m*	the most significant bits(MSB) of a 16-bit constant
mn	*mn*	16-bit constant
n	*n*	8-bit constant or the least significant bits(LSB) of a 16-bit constant
r, r'	*g, g'*	byte register select (000 = B, 001 = C, 010 = D, 011 = E, 100 = H, 101 = L, 111 = A)
ss	*ww*	word register select-source (00 = BC, 01 = DE, 10 = HL, 11 = SP)
v	*v*	Restart address select (010 = 0020h, 011 = 0030h, 100 = 0040h, 101 = 0050h, 111 = 0070h)
x	*nbr*	an 8-bit constant to load into the XPC
xx	*xx*	word register select (00 = BC, 01 = DE, 10 = IX, 11 = SP)
yy	*yy*	word register select (00 = BC, 01 = DE, 10 = IY, 11 = SP)
zz	*zz*	word register select (00 = BC, 01 = DE, 10 = HL, 11 = AF)

Condition Codes

Table 6: Condition Code Description

Condition	Flag=Value	Description
NZ	Z=0	Not Zer0
Z	Z=1	Zero
NC	C=0	No Carry (C=0)
C	C=1	Carry (C=1)
P	S=0	Minus
M	S=1	Positive
LZ	L/V=0	For logic operations, Logic Zero (all of the four most significant bits of the result are zero)
NV	L/V=0	For arithmentic operations, No Overflow
LO	L/V=1	For logic operations, Logic One (one or more of the four most signif-icant bits of the result are one)
V	L/V=1	For arithmentic operations, Overflow

3. Processor Registers



4. OpCode Descriptions

ADC A,(HL)
ADC A,(IX+*d*)
ADC A,(IY+*d*)

Opcode	Instruction	Clocks	Operation
8E	ADC A,(HL)	5 (2,1,2)	A= A + (HL) + CF
DD 8E *d*	ADC A,(IX+*d*)	9 (2,2,2,1,2)	A= A + (IX+*d*) + CF
FD 8E *d*	ADC A,(IY+*d*)	9 (2,2,2,1,2)	A= A + (IY+*d*) + CF

Flags							
S	Z				L/V		C
•	•				V		•

ALTD		
F	R	SP
•	•	

I/O	
S	D
•	

Description

The data in the Accumulator is summed with the Carry Flag and with the data in memory whose location is:

- held in word register HL, or
- the sum of the data in index register IX and a displacement value *d*, or
- the sum of the data in index register IY and a displacement value *d*.

The result is then stored in the Accumulator.

ADC A,*n*

Opcode	Instruction	Clocks	Operation
CE *n*	ADC A,*n*	4 (2,2)	A = A + *n* + CF

Flags							
S	Z				L/V		C
•	•				V		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

The 8-bit constant *n* is summed with the Carry Flag and with the data in the Accumulator. The sum is then stored in the Accumulator.

ADC A,*r*

Opcode	Instruction	Clocks	Operation
——	**ADC A,*r***	**2**	**A = A + *r* + CF**
8F	ADC A,A	2	A = A + A + CF
88	ADC A,B	2	A = A + B + CF
89	ADC A,C	2	A = A + C + CF
8A	ADC A,D	2	A = A + D + CF
8B	ADC A,E	2	A = A + E + CF
8C	ADC A,H	2	A = A + H + CF
8D	ADC A,L	2	A = A + L + CF

Flags							
S	Z				L/V		C
•	•				V		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

The data in the Accumulator is summed with the Carry Flag, CF, and with the data in register *r* (any of the registers A, B, C, D, E, H, or L). The result is stored in the Accumulator.

ADC HL,*ss*

Opcode	Instruction	Clocks	Operation
——	**ADC HL,*ss***	**4 (2,2)**	**HL = HL + *ss* + CF**
ED 4A	ADC HL,BC	4 (2,2)	HL = HL + BC + CF
ED 5A	ADC HL,DE	4 (2,2)	HL = HL + DE + CF
ED 6A	ADC HL,HL	4 (2,2)	HL = HL + HL + CF
ED 7A	ADC HL,SP	4 (2,2)	HL = HL + SP + CF

Flags							
S	Z				L/V		C
•	•				V		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

The data in the register pair HL is summed with the Carry Flag and with the data in word register *ss* (any of the word registers BC, DE, HL, or SP). The result is stored in HL.

ADD A,(HL)
ADD A,(IX+*d*)
ADD A,(IY+*d*)

Opcode	Instruction	Clocks	Operation
86	ADD A,(HL)	5 (2,1,2)	A = A + (HL)
DD 86 *d*	ADD A,(IX+*d*)	9 (2,2,2,1,2)	A = A + (IX+*d*)
FD 86 *d*	ADD A,(IY+*d*)	9 (2,2,2,1,2)	A = A + (IY+*d*)

Flags							
S	Z				L/V		C
•	•				V		•

ALTD		
F	R	SP
•	•	

I/O	
S	D
•	

Description

The data in the Accumulator is summed with the data in the memory location whose address is:

- held in word register HL, or
- the sum of the data in index register IX and a displacement value *d*, or
- the sum of the data in index register IY and a displacement value *d*.

The result is stored in the Accumulator.

ADD A,*n*

Opcode	Instruction	Clocks	Operation
C6 *n*	ADD A,*n*	4 (2,2)	A = A + *n*

Flags							
S	Z				L/V		C
•	•				V		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

The data in the Accumulator is summed with the 8-bit constant *n*. The result is stored in the Accumulator.

ADD A,*r*

Opcode	Instruction	Clocks	Operation
——	**ADD A,*r***	**2**	**A = A + *r***
87	ADD A,A	2	A = A + A
80	ADD A,B	2	A = A + B
81	ADD A,C	2	A = A + C
82	ADD A,D	2	A = A + D
83	ADD A,E	2	A = A + E
84	ADD A,H	2	A = A + H
85	ADD A,L	2	A = A + L

Flags							
S	Z				L/V		C
•	•				V		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

The data in the Accumulator is summed with the data in register *r* (any of the registers A, B, C, D, E, H, or L). The result is stored in the Accumulator.

ADD HL,*ss*

Opcode	Instruction	Clocks	Operation
——	**ADD HL,*ss***	**2**	**HL = HL + *ss***
09	ADD HL,BC	2	HL = HL + BC
19	ADD HL,DE	2	HL = HL + DE
29	ADD HL,HL	2	HL = HL + HL
39	ADD HL,SP	2	HL = HL + SP

Flags							
S	Z				L/V		C
-	-				-		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

The data in the word register HL is summed with the data in the word register *ss* (any of the word registers BC, DE, HL, or SP). The result is stored in HL.

ADD IX,*xx*
ADD IY,*yy*

Opcode	Instruction	Clocks	Operation
——	**ADD IX,*xx***	**4 (2,2)**	**IX = IX + *xx***
DD 09	ADD IX,BC	4 (2,2)	IX = IX + BC
DD 19	ADD IX,DE	4 (2,2)	IX = IX + DE
DD 29	ADD IX,IX	4 (2,2)	IX = IX + IX
DD 39	ADD IX,SP	4 (2,2)	IX = IX + SP
——	**ADD IY,*yy***	**4 (2,2)**	**IY = IY + *yy***
FD 09	ADD IY,BC	4 (2,2)	IY = IY + BC
FD 19	ADD IY,DE	4 (2,2)	IY = IY + DE
FD 29	ADD IY,IY	4 (2,2)	IY = IY + IY
FD 39	ADD IY,SP	4 (2,2)	IY = IY + SP

Flags							
S	Z				L/V		C
-	-				-		•

ALTD		
F	R	SP
•		

I/O	
S	D

Description

The data in index register IX is summed with the word register *xx* (any of the word registers BC, DE, IX, or SP) and the result is stored in IX.

The data in index register IY is summed with the word register *yy* (any of the word registers BC, DE, IY, or SP). The result is stored in IY.

ADD SP,*d* N

Opcode	Instruction	Clocks	Operation
27 *d*	ADD SP,*d*	4 (2,2)	SP = SP + *d*

Flags							
S	Z						C
-	-				-		•

ALTD		
F	R	SP
•		

I/O	
S	D

Description

The data in the Stack Pointer register is summed with a displacement value *d*, and then stored in SP.

ALTD N

Opcode	Instruction	Clocks	Operation
76	ALTD	2	[Sets alternate register destination for following instruction.]

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

This is an instruction prefix. Causes the instruction immediately following to affect the alternate flags, or use the alternate registers for the destination of the data, or both. For some instructions ALTD causes special alternate register uses, unique to that instruction.

Example

The instruction

```
ALTD ADD HL,DE
```

would add the data in word register DE to the data in word register HL and store the result in the alternate word register HL'.

The instructions

```
ALTD LD DE,BC
```

and

```
LD DE',BC
```

both load the data in word register BC into the alternate word register DE'.

Description

Performs a logical AND operation between the byte in the Accumulator and the byte whose address is:

- in word registerss

AND HL,DE N

Opcode	Instruction	Clocks	Operation
DC	AND HL,DE	2	HL = HL& DE

Flags							
S	Z				L/V		C
•	•				L		0

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Performs a logical AND operation between the word in word register HL and the word in word register DE. The relative bits of each byte are compared (i.e., the bit 1 of both bytes are compared, the bit 2 of both bytes are compared, etc.) and the associated bit in the result byte is set only if both the compared bits are set. The result is stored in HL.

AND IX,DE N
AND IY,DE N

Opcode	Instruction	Clocks	Operation
DD DC	AND IX,DE	4 (2,2)	IX = IX & DE
FD DC	AND IY,DE	4 (2,2)	IY = IY & DE

Flags							
S	Z				L/V		C
•	•				L		0

ALTD		
F	R	SP
•		

I/O	
S	D

Description

- **AND IX,DE** performs a logical AND operation between the word in index register IX and the word in word register DE. The result is stored in IX.
- **AND IY,DE** performs a logical AND operation between the word in index register IY and the word in word register DE. The result is stored in IY.

The relative bits of each byte are compared (i.e., the bit 1 of both bytes are compared, the bit 2 of both bytes are compared, etc.) and the associated bit in the result byte is set only if both the compared bits are set.

AND *n*

Opcode	Instruction	Clocks	Operation
E6 *n*	AND *n*	4 (2,2)	A = A & *n*

Flags							
S	Z				L/V		C
•	•				L		0

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Performs a logical AND operation between the byte in the Accumulator and the 8-bit constant *n*. The relative bits of each byte are compared (i.e., the bit 1 of both bytes are compared, the bit 2 of both bytes are compared, etc.) and the associated bit in the result byte is set only if both the compared bits are set. The result is stored in the Accumulator.

AND *r*

Opcode	Instruction	Clocks	Operation
——	**AND *r***	**2**	**A = A & *r***
A7	AND A	2	A = A & A
A0	AND B	2	A = A & B
A1	AND C	2	A = A & C
A2	AND D	2	A = A & D
A3	AND E	2	A = A & E
A4	AND H	2	A = A & H
A5	AND L	2	A = A & L

Flags							
S	Z				L/V		C
•	•				L		0

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Performs a logical AND operation between the byte in the Accumulator and the byte in the register *r* (any of the registers A, B, C, D, E, H, or L). The relative bits of each byte are compared (i.e., the bit 1 of both bytes are compared, the bit 2 of both bytes are compared, etc.) and the associated bit in the result byte is set only if both the compared bits are set. The result is stored in the Accumulator.

BIT *b*,(HL) P
BIT *b*,(IX+*d*)
BIT *b*,(IY+*d*)

Opcode	Instruction	Clocks	Operation
——	**BIT *b*,(HL)**	**7 (2,2,1,2)**	**(HL) & bit**
CB 46	BIT 0,(HL)	7 (2,2,1,2)	(HL) & bit 0
CB 4E	BIT 1,(HL)	7 (2,2,1,2)	(HL) & bit 1
CB 56	BIT 2,(HL)	7 (2,2,1,2)	(HL) & bit 2
CB 5E	BIT 3,(HL)	7 (2,2,1,2)	(HL) & bit 3
CB 66	BIT 4,(HL)	7 (2,2,1,2)	(HL) & bit 4
CB 6E	BIT 5,(HL)	7 (2,2,1,2)	(HL) & bit 5
CB 76	BIT 6,(HL)	7 (2,2,1,2)	(HL) & bit 6
CB 7E	BIT 7,(HL)	7 (2,2,1,2)	(HL) & bit 7
——	**BIT *b*,(IX+*d*)**	**10 (2,2,2,2,2)**	**(IX+*d*) & bit**
DD CB *d* 46	BIT 0,(IX+*d*)	10 (2,2,2,2,2)	(IX+*d*) & bit 0
DD CB *d* 4E	BIT 1,(IX+*d*)	10 (2,2,2,2,2)	(IX+*d*) & bit 1
DD CB *d* 56	BIT 2,(IX+*d*)	10 (2,2,2,2,2)	(IX+*d*) & bit 2
DD CB *d* 5E	BIT 3,(IX+*d*)	10 (2,2,2,2,2)	(IX+*d*) & bit 3
DD CB *d* 66	BIT 4,(IX+*d*)	10 (2,2,2,2,2)	(IX+*d*) & bit 4
DD CB *d* 6E	BIT 5,(IX+*d*)	10 (2,2,2,2,2)	(IX+*d*) & bit 5
DD CB *d* 76	BIT 6,(IX+*d*)	10 (2,2,2,2,2)	(IX+*d*) & bit 6
DD CB *d* 7E	BIT 7,(IX+*d*)	10 (2,2,2,2,2)	(IX+*d*) & bit 7
——	**BIT *b*,(IY+*d*)**	**10 (2,2,2,2,2)**	**(IY+*d*) & bit**
FD CB *d* 46	BIT 0,(IY+*d*)	10 (2,2,2,2,2)	(IY+*d*) & bit 0
FD CB *d* 4E	BIT 1,(IY+*d*)	10 (2,2,2,2,2)	(IY+*d*) & bit 1
FD CB *d* 56	BIT 2,(IY+*d*)	10 (2,2,2,2,2)	(IY+*d*) & bit 2
FD CB *d* 5E	BIT 3,(IY+*d*)	10 (2,2,2,2,2)	(IY+*d*) & bit 3
FD CB *d* 66	BIT 4,(IY+*d*)	10 (2,2,2,2,2)	(IY+*d*) & bit 4
FD CB *d* 6E	BIT 5,(IY+*d*)	10 (2,2,2,2,2)	(IY+*d*) & bit 5
FD CB *d* 76	BIT 6,(IY+*d*)	10 (2,2,2,2,2)	(IY+*d*) & bit 6
FD CB *d* 7E	BIT 7,(IY+*d*)	10 (2,2,2,2,2)	(IY+*d*) & bit 7

Flags							
S	Z				L/V		C
-	•				-		-

ALTD		
F	R	SP
•		

I/O	
S	D
•	

Description

Tests the bit *b* (any of the bits 0, 1, 2, 3, 4, 5, 6, or 7) of the byte whose address is:

- contained in the register pair HL, or
- the sum of data in index register IX plus a displacement value *d*, or
- the data in index register IY plus a displacement value *d*.

The Zero Flag, Z, is set if the tested bit is 0, reset the bit is 1.

BIT *b,r*

Opcode								Instruction	Clocks	Operation
								BIT *b,r*	4 (2,2)	*r* & bit
b,r	A	B	C	D	E	H	L			
CB (0)	47	40	41	42	43	44	45			
CB (1)	4F	48	49	4A	4B	4C	4D			
CB (2)	57	50	51	52	53	54	55			
CB (3)	5F	58	59	5A	5B	5C	5D			
CB (4)	67	60	61	62	63	64	65			
CB (5)	6F	68	69	6A	6B	6C	6D			
CB (6)	77	70	71	72	73	74	75			
CB (7)	7F	78	79	7A	7B	7C	7D			

Flags							
S	Z				L/V		C
-	•				-		-

ALTD		
F	R	SP
•		

I/O	
S	D

Description

Tests the bit *b* (any of the bits 0, 1, 2, 3, 4, 5, 6, or 7) of the byte in the register *r* (any of the registers A, B, C, D, E, H, or L).

The Zero Flag, Z, is set if the tested bit is 0, reset if the bit is 1.

BOOL HL N

Opcode	Instruction	Clocks	Operation
CC	BOOL HL	2	If (HL != 0) HL = 1

Flags							
S	Z				L/V		C
•	•				0		0

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

If the data in word register HL does not equal zero, then it is set to 1.

BOOL IX N
BOOL IY N

Opcode	Instruction	Clocks	Operation
DD CC	BOOL IX	4 (2,2)	If (IX != 0) IX = 1
FD CC	BOOL IY	4 (2,2)	If (IY != 0) IY = 1

Flags							
S	Z				L/V		C
•	•				0		0

ALTD		
F	R	SP
•		

I/O	
S	D

Description

If the data in index register IX or IY does not equal zero, then that register is set to 1.

CALL *mn*

Opcode	Instruction	Clocks	Operation
CD *n m*	CALL *mn*	12 (2,2,2,3,3)	(SP - 1) = $PC_{(high)}$; (SP - 2) = $PC_{(low)}$; PC = *mn*; SP = SP - 2

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

This instruction is used to call a subroutine. First the data in the Program Counter is pushed onto the stack. The high-order byte of the PC is pushed first, then the low-order byte. The program counter is then loaded with *mn*,16-bit address of the first instruction of the subroutine. The Stack Pointer is updated to reflect the two bytes pushed onto the stack.

The Dynamic C assembler recognizes **CALL *label***, where ***mn*** is coded as a label.

CCF

Opcode	Instruction	Clocks	Operation
3F	CCF	2	CF = ~CF

Flags							
S	Z				L/V		C
-	-				-		•

ALTD		
F	R	SP
•		

I/O	
S	D

Description

The Carry Flag is inverted: If it is set, it becomes cleared. If it is not set, it becomes set.

CP (HL)
CP (IX+*d*)
CP (IY+*d*)

Opcode	Instruction	Clocks	Operation
BE	CP (HL)	5 (2,1,2)	A - (HL)
DD BE *d*	CP (IX + *d*)	9 (2,2,2,1,2)	A - (IX + *d*)
FE BE *d*	CP (IY + *d*)	9 (2,2,2,1,2)	A - (IY + *d*)

Flags							
S	Z				L/V		C
•	•				V		•

ALTD		
F	R	SP
•		

I/O	
S	D
•	

Description

Compares the data in the Accumulator with the data whose address is (a) contained in word register HL, (b) the sum of the data in index register IX plus a displacement value *d*, or (c) the sum of the data in index register IY plus a displacement value *d*.

These compares are accomplished by subtracting the appropriate data ((HL), (IX+*d*), or (IY+*d*)) from the Accumulator. If the value of the data in the Accumulator is less than the value of the data compared, then the Sign Flag and the Carry Flag are set. If they are equal, the Zero Flag is set. If the data is greater than the data in the Accumulator, then the Sign, Carry, and Zero Flags are reset. This operation does not affect the data in the Accumulator.

CP *n*

Opcode	Instruction	Clocks	Operation
FE *n*	CP *n*	4 (2,2)	A - *n*

Flags							
S	Z				L/V		C
•	•				V		•

ALTD		
F	R	SP
•		

I/O	
S	D

Description

Compares the data in the Accumulator with an 8-bit constant *n*. This compare is accomplished by subtracting *n* from the Accumulator. If the value of the data in the Accumulator is less than the value of *n*, then the Sign Flag and the Carry Flag are set. If they are equal, the Zero Flag is set. If *n* is greater than the data in the Accumulator, then the Sign, Carry, and Zero Flags are reset. This operation does not affect the data in the Accumulator.

CP *r*

Opcode	Instruction	Clocks	Operation
—	**CP *r***	**2**	**A - *r***
BF	CP A	2	A - A
B8	CP B	2	A - B
B9	CP C	2	A - C
BA	CP D	2	A - D
BB	CP E	2	A - E
BC	CP H	2	A - H
BD	CP L	2	A - L

Flags							
S	Z				L/V		C
•	•				V		•

ALTD		
F	R	SP
•		

I/O	
S	D

Description

Compares the data in Accumulator with the data in register *r* (any of the registers A, B, C, D, E, H, or L). This compare is accomplished by subtracting the appropriate data (*r*) from the Accumulator. If the value of the data in the Accumulator is less than the value of the data compared, then the Sign Flag and the Carry Flag are set. If they are equal, the Zero Flag is set. If the data is greater than the data in the Accumulator, then the Sign, Carry, and Zero Flags are reset. This operation does not affect the data in the Accumulator.

CPL

Opcode	Instruction	Clocks	Operation
2F	CPL	2	A = ~A

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP
	•	

I/O	
S	D

Description

The data in the Accumulator is inverted (one's complement).

Example

If the data in the Accumulator is 1100 0101, after the instruction CPL the Accumulator will contain 0011 1010.

DEC (HL)
DEC (IX+*d*)
DEC (IY+*d*)

Opcode	Instruction	Clocks	Operation
35	DEC (HL)	8 (2,1,2,3)	(HL) = (HL) - 1
DD 35 *d*	DEC (IX+D)	12 (2,2,2,1,2,3)	(IX + *d*) = (IX + *d*) - 1
FD 35 *d*	DEC (IY+D)	12 (2,2,2,1,2,3)	(IY + *d*) = (IY + *d*) - 1

Flags							
S	Z				L/V		C
•	•				V		-

ALTD		
F	R	SP
•		

I/O	
S	D
•	•

Description

Decrements the byte whose address is:

- in word register HL, or
- the data in index register IX plus a displacement value *d*, or
- the data in index register IY plus a displacement value *d*.

DEC IX
DEC IY

Opcode	Instruction	Clocks	Operation
DD 2B	DEC IX	4 (2,2)	IX = IX - 1
FD 2B	DEC IY	4(2,2)	IY = IY - 1

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

Decrements the data in index register IX or IY.

DEC *r*

Opcode	Instruction	Clocks	Operation
——	**DEC *r***	**2**	***r* = *r* - 1**
3D	DEC A	2	A = A - 1
05	DEC B	2	B = B - 1
0D	DEC C	2	C = C - 1
15	DEC D	2	D = D - 1
1D	DEC E	2	E = E - 1
25	DEC H	2	H = H - 1
2D	DEC L	2	L = L - 1

Flags							
S	Z				L/V		C
•	•				V		-

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Decrements the data in the register *r* (any of the registers A, B, C, D, E, H, or L).

DEC *ss*

Opcode	Instruction	Clocks	Operation
——	**DEC *ss***	**2**	***ss* = *ss* - 1**
0D	DEC BC	2	BC = BC - 1
1D	DEC DE	2	DE = DE - 1
2D	DEC HL	2	HL = HL - 1
3D	DEC SP	2	SP = SP - 1

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP
	•	

I/O	
S	D

Description

Decrements the data in word register *ss* (any of the word registers BC, DE, HL, or SP).

DJNZ *e*

Opcode	Instruction	Clocks	Operation
10 *e*-2	DJNZ *e*	5 (2,2,1)	B = B-1; if {B != 0} PC = PC + *e*

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP
	•	

I/O	
S	D

Description

This instruction’s mnemonic stands for Decrement and Jump if Not Zero. It decrements the data in register B then, if the data in B does not equal 0, it adds the 8-bit signed constant *e* to the Program Counter.

Two is subtracted from the value *e* so the instruction jumps from the current instruction and not the following instruction.

EX (SP),HL M

Opcode	Instruction	Clocks	Operation
ED 54	EX (SP),HL	15 (2,2,1,2,2,3,3)	H <–> (SP+1); L <–> (SP)

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP
	•	

I/O	
S	D

Description

Exchanges the byte in the register H with the data whose address is the data in the Stack Pointer register plus 1; and exchanges the byte in the register L with the data whose address is the data in the Stack Pointer.

EX (SP),IX
EX (SP),IY

Opcode	Instruction	Clocks	Operation
DD E3	EX (SP),IX	15 (2,2,1,2,2,3,3)	$IX_{(high)}$ <–> (SP+1); $IX_{(low)}$ <–> (SP)
FD E3	EX (SP),IY	15 (2,2,1,2,2,3,3)	$IY_{(high)}$ <–> (SP+1); $IY_{(low)}$ <–> (SP)

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

- **EX (SP),IX** exchanges the high order byte of index register IX with the data whose address is 1 plus the data in the Stack Pointer register, and exchanges the low order byte of index register IX with the data whose address is the data in the Stack Pointer register, SP.
- **EX (SP),IY** exchanges the high order byte of index register IY with the data whose address is 1 plus the data in the Stack Pointer register, and exchanges the low order byte of index register IY with the data whose address is the data in the Stack Pointer register.

EX AF,AF'

Opcode	Instruction	Clocks	Operation
08	EX AF,AF'	2	AF <–> AF'

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

Exchanges the data in word register AF with the data in the alternate word register AF'.

EX DE,HL
EX DE',HL

N

Opcode	Instruction	Clocks	Operation
EB	EX DE,HL	2	if (!ALTD) then DE <–> HL else DE <–> HL'
E3	EX DE',HL	2	if (!ALTD) then DE' <–> HL else DE' <–> HL'

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP
		•

I/O	
S	D

Description

- **EX DE,HL** exchanges the data in word register DE with the data in word register HL. If the ALTD instruction is present then the data in DE is exchanged with the data in the alternate word register HL'.
- **EX DE',HL** exchanges the data in the alternate word register DE' with the data in word register HL. If the ALTD instruction is present then the data in DE' is exchanged with the data in the alternate word register HL'.

The Dynamic C assembler recognizes the following instructions, which are based on a combination of ALTD and the above exchange operations:

- **EX DE’,HL’ ; equivalent to ALTD EX DE’,HL**
- **EX DE,HL’ ; equivalent to ALTD EX DE’,HL’**

EXX

Opcode	Instruction	Clocks	Operation
D9	EXX	2	BC <–> BC'; DE <–> DE'; HL <–> HL'

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

Exchanges the data in word registers BC, DE, and HL, with the data in their respective alternate word registers BC', DE', and HL'.

INC (HL)
INC (IX+*d*)
INC (IY+*d*)

Opcode	Instruction	Clocks	Operation
34	INC (HL)	8 (2,1,2,3)	(HL) = (HL) + 1
DD 34 *d*	INC (IX+*d*)	12 (2,2,2,1,2,3)	(IX + *d*) = (IX + *d*) + 1
FD 34 *d*	INC (IY+*d*)	12 (2,2,2,1,2,3)	(IY + *d*) = (IY + *d*) + 1

Flags							
S	Z				L/V		C
•	•				V		-

ALTD		
F	R	SP
•		

I/O	
S	D
•	•

Description

Increments the byte whose address is:

- held in word register HL, or
- the sum of the data in index register IX and a displacement value *d*, or
- the sum of the data in index register IY and a displacement value *d*.

INC IX
INC IY

Opcode	Instruction	Clocks	Operation
DD 23	INC IX	4 (2,2)	IX = IX + 1
FD 23	INC IY	4 (2,2)	IY = IY + 1

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

- **INC IX** increments the data in index register IX.
- **INC IY** increments the data in index register IY.

INC *r*

Opcode	Instruction	Clocks	Operation
——	**INC *r***	**2**	***r* = *r* + 1**
3C	INC A	2	A = A + 1
04	INC B	2	B = B + 1
0C	INC C	2	C = C + 1
14	INC D	2	D = D + 1
1C	INC E	2	E = E + 1
24	INC H	2	H = H + 1
2C	INC L	2	L = L + 1

Flags							
S	Z				L/V		C
•	•				V		-

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Increments the data in the register *r* (any of the registers A, B, C, D, E, H, or L).

INC *ss*

Opcode	Instruction	Clocks	Operation
——	**INC *ss***	**2**	***ss* = *ss* + 1**
03	INC BC	2	BC = BC + 1
13	INC DE	2	DE = DE + 1
23	INC HL	2	HL = HL + 1
33	INC SP	2	SP = SP + 1

Flags							
S	**Z**				**L/V**		**C**
-	-				-		-

ALTD		
F	**R**	**SP**
	•	

I/O	
S	**D**

Description

Increments the data in word register *ss* (any of the word registers BC, DE, HL, or SP).

IOE N
IOI N

Opcode	Instruction	Clocks	Operation
DD	IOE	2	I/O external prefix
D3	IOI	2	I/O internal prefix

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

- **IOI:** The IOI prefix allows the use of existing memory access instructions as internal I/O instructions. When prefixed, a 16-bit memory instruction accesses the I/O space at the address specified by the lower byte of the 16-bit address. With IOI, the upper byte of a 16-bit address is ignored since internal I/O peripherals are mapped within the first 256-bytes of the I/O address space. Writes to inetnal I/O registers require two clocks rather than the three required for memory write operations.
- **IOE:** The IOE prefix allows the use of existing memory access instructions as external I/O instructions. Unlike internal I/O peripherals, external I/O devices can be mapped within 8K of the available 64K address space. Therefore, prefixed 16-bit memory access instructions can be used more appropriately for external I/O operations. By default, writes are inhibited for external I/O operations and fifteen wait states are added for I/O accesses.

WARNING: If an I/O prefixed instruction is immediately followed by one of these 12 special one byte memory access instructions, a bug in the Rabbit 2000 causes I/O access to occur instead of memory access:

```
ADC A,(HL)      CP (HL)         SUB (HL)        INC (HL)
ADD A,(HL)      OR (HL)         XOR (HL)        LD r,(HL)
AND (HL)        SBC A,(HL)      DEC (HL)        LD (HL),r
```

This bug can easily be avoided by putting a **NOP** instruction between an I/O instruction and any of these special instructions. Dynamic C versions 6.57 and later will automatically compensate for this bug.

Examples

The following instruction loads the contents of the Accumulator into the internal I/O register at address location 030h:

```
IOI  LD (030h), A
```

These next instructions read a word from external I/O address 0A002:

```
LD  IX, 0A000h
IOE  LD HL, (IX+2)
```

IP 0 NP
IP 1 NP
IP 2 NP
IP 3 NP

Opcode	Instruction	Clocks	Operation
ED 46	IP 0	4 (2,2)	IP = {IP[5:0], 00}
ED 56	IP 1	4 (2,2)	IP = {IP[5:0], 01}
ED 4E	IP 2	4 (2,2)	IP = {IP[5:0], 10}
ED 5E	IP 3	4 (2,2)	IP = {IP[5:0], 11}

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

The Interrupt Priority Register, IP is an 8-bit register that forms a stack of the current priority and the other previous 3 priorities. IP 0 forms the lowest priority; IP 3 forms the highest priority.

- **IP 0:** The IP 0 instruction shifts the contents of the register holding the previous priorities 2-bits to the left, then sets the Interrupt Priority Register (bits 0 and 1) to 00.
- **IP 1:** The IP 1 instruction first shifts the contents of the register holding the previous priorities 2-bits to the left, then sets the Interrupt Priority Register (bits 0 and 1) to 01.
- **IP 2:** The IP 2 instruction shifts the contents of the register holding the previous priorities 2-bits to the left, then sets the Interrupt Priority Register (bits 0 and 1) to 10.
- **IP 3:** The IP 3 instruction shifts the contents of the register holding the previous priorities 2-bits to the left, then sets the Interrupt Priority Register (bits 0 and 1) to 11.

Processor Priority	Effect on Interrupts
0	All interrupts, priority 1,2 and 3 take place after execution of current non privileged instruction.
1	Only interrupts of priority 2 and 3 take place after execution of current non privileged instruction.
2	Only interrupts of priority 3 take place after execution of current non privileged instruction.
3	All interrupt are suppressed (except the RST instruction).

IPRES NP

Opcode	Instruction	Clocks	Operation
ED 5D	IPRES	4 (2,2)	IP = {IP[1:0], IP[7:2]}

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

The IPRES instruction rotates the contents of the Interrupt Priority Register 2-bits to the right, replacing the current priority with the previous priority. It is impossible to interrupt during the execution of this instruction.

Example

If the Interrupt Priority register contains 00000110, the execution of the instruction

```
IPRES
```

would cause the Interrupt Priority register to contain 10000001.

JP (HL)
JP (IX)
JP (IY)
JP *mn*

Opcode	Instruction	Clocks	Operation
E9	JP (HL)	4 (2,2)	PC = HL
DD E9	JP (IX)	6 (2,2,2)	PC = IX
FD E9	JP (IY)	6 (2,2,2)	PC = IY
C3 *n m*	JP *mn*	7 (2,2,2,1)	PC = *mn*

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

- **JP (HL):** The data in HL is loaded into the Program Counter. Thus the address of the next instruction fetched is the data in HL.
- **JP (IX):** The data in index register IX is loaded into the Program Counter. Thus the address of the next instruction fetched is the data in IX.
- **JP (IY):** The data in index register IY is loaded into the Program Counter. Thus the address of the next instruction fetched is the data in IY.
- **JP *mn*:** The 16-bit constant *mn* is loaded into the Program Counter. Thus the address of the next instruction fetched is *mn*. This instruction recognizes labels when used in the Dynamic C assembler.

JP *f,mn*

Opcode	Instruction	Clocks	Operation
——	**JP *f,mn***	**7 (2,2,2,1)**	**if {*f*} PC = *mn***
C2 *n m*	JP NZ,*mn*	7 (2,2,2,1)	if {NZ} PC = *mn*
CA *n m*	JP Z,*mn*	7 (2,2,2,1)	if {Z} PC = *mn*
D2 *n m*	JP NC,*mn*	7 (2,2,2,1)	if {NC} PC = *mn*
DA *n m*	JP C,*mn*	7 (2,2,2,1)	if {C} PC = *mn*
E2 *n m*	JP LZ,*mn*	7 (2,2,2,1)	if {LZ/NV} PC = *mn*
EA *n m*	JP LO,*mn*	7 (2,2,2,1)	if {LO/V} PC = *mn*
F2 *n m*	JP P,*mn*	7 (2,2,2,1)	if {P} PC = *mn*
FA *n m*	JP M,*mn*	7 (2,2,2,1)	if {M} PC = *mn*

Flags							
S	**Z**				**L/V**		**C**
-	-				-		-

ALTD		
F	**R**	**SP**

I/O	
S	**D**

Description

If the condition *f* is true then the 16-bit data *mn* is loaded into the Program Counter, PC. If the condition is false then the Program Counter increments normally.

The condition *f* is one of the following: NZ, zero flag not set; Z, zero flag set; NC, carry flag not set; C, carry flag set; LZ, Logical/Overflow flag is not set; LO, Logical/Overflow flag is set; P, sign flag not set; M, sign flag set.

This instruction recognizes labels when used in the Dynamic C assembler.

JR *cc,e*

Opcode	Instruction	Clocks	Operation
——	**JR *cc,e***	**5 (2,2,1)**	**if {*cc*} PC = PC + *e***
20 *e*-2	JR NZ,*e*	5 (2,2,1)	if {NZ} PC = PC + *e*
28 *e*-2	JR Z,*e*	5 (2,2,1)	if {Z} PC = PC + *e*
30 *e*-2	JR NC,*e*	5 (2,2,1)	if {NC} PC = PC + *e*
38 *e*-2	JR C,*e*	5 (2,2,1)	if {C} PC = PC + *e*

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

If condition *cc* is true then the 8-bit signed displacement value *e* is added to the Program Counter, PC.

Since the instruction takes two increments of the PC to complete, two is subtracted from the displacement value so that the displacement take place from the instruction opcode.

This instruction recognizes labels when used in the Dynamic C assembler.

JR *e*

Opcode	Instruction	Clocks	Operation
18 *e*-2	JR *e*	5 (2,2,1)	PC = PC + *e*

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

Adds a signed constant *e* to the Program Counter.

Since the instruction takes two increments of the PC to complete, two is subtracted from the displacement value so that the displacement take place from the instruction opcode.

This instruction recognizes labels when used in the Dynamic C assembler.

LCALL *x,mn* N

Opcode	Instruction	Clocks	Operation
CF *n m x*	LCALL *x,mn*	19 (2,2,2,2,1,3,3,3,1)	(SP - 1) = XPC; (SP - 2) = $PC_{(high)}$; (SP - 3) = $PC_{(low)}$; XPC = *x*; PC = *mn*; SP = SP - 3

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

This instruction is similar to the CALL routine in that it transfers program execution to the subroutine address specified by the 16-bit operand *mn*. The LCALL instruction is special in that it allows calls to be made to a computed address in XMEM. Note that the value of XPC and consequently the address space defined by the XPC is dynamically changed with the LCALL instructions.

In the LCALL instruction, first the Extension of the Program Counter, XPC, is pushed onto the stack. Next the Program Counter, PC, is pushed onto the stack, the high order byte first, then the low order byte. Then the XPC is loaded with the 8-bit value *x* and the PC is loaded with the 16-bit value, *mn*. The Stack Pointer register is then updated to reflect the three items pushed onto it.

The value *mn* must be in the range E000–FFFF.

Alternate Forms

The Dynamic C assembler recognizes several other forms of this instruction.

LCALL *label*

LCALL *x,label*

LCALL *x*:*label*

LCALL *x*:*mn*

The parameter *label* is a user defined label. The colon is equivalent to the comma as a delimiter.

LD (BC),A
LD (DE),A
LD (HL),*n*
LD (HL),*r*

Opcode	Instruction	Clocks	Operation
02	LD (BC),A	7 (2,2,3)	(BC) = A
12	LD (DE),A	7 (2,2,3)	(DE) = A
36 *n*	LD (HL),*n*	7 (2,2,3)	(HL) = *n*

——
77
70
71
72
73
74
75
	LD (HL),*r*
	LD (HL),A
	LD (HL),B
	LD (HL),C
	LD (HL),D
	LD (HL),E
	LD (HL),H
	LD (HL),L
		6 (2,1,3)
		6 (2,1,3)
		6 (2,1,3)
		6 (2,1,3)
		6 (2,1,3)
		6 (2,1,3)
		6 (2,1,3)
		6 (2,1,3)
			(HL) = *r*
			(HL) = A
			(HL) = B
			(HL) = C
			(HL) = D
			(HL) = E
			(HL) = H
			(HL) = L

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D
	•

Description

- **LD (BC),A:** Loads the memory location whose address is the data in word register BC with the data in the Accumulator.
- **LD (DE),A:** Loads the memory location whose address is the data in word register DE with the data in the Accumulator.
- **LD (HL),*n*:** Loads the memory location whose address is the data in HL with the 8-bit constant *n*.
- **LD (HL),*r*:** Loads the memory location whose address is the data in HL, with the data in the register *r* (any of the registers A, B, C, D, E, H, or L).

LD (HL+*d*),HL N

Opcode	Instruction	Clocks	Operation
DD F4 *d*	LD (HL+*d*),HL	13 (2,2,2,1,3,3)	(HL+*d*) = L; (HL+*d*+1) = H

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D
	•

Description

Loads the data in register L into the memory location whose address is the sum of the data in word register HL and a displacement value *d*. Then, loads the data in register H into the memory location whose address is the sum of the data in word register HL and a displacement value *d* plus 1.

LD (IX+*d*),HL
LD (IX+*d*),*n*
LD (IX+*d*),*r*

N

Opcode	Instruction	Clocks	Operation
F4 *d*	LD (IX+*d*),HL	11 (2,2,1,3,3)	(IX + *d*) = L; (IX + *d* + 1) = H
DD 36 *d n*	LD (IX+*d*),*n*	11 (2,2,2,2,3)	(IX + *d*) = *n*
——	**LD (IX+*d*),*r***	**10 (2,2,2,1,3)**	**(IX + *d*) = *r***
DD 77 *d*	LD (IX+*d*),A	10 (2,2,2,1,3)	(IX + *d*) = A
DD 70 *d*	LD (IX+*d*),B	10 (2,2,2,1,3)	(IX + *d*) = B
DD 71 *d*	LD (IX+*d*),C	10 (2,2,2,1,3)	(IX + *d*) = C
DD 72 *d*	LD (IX+*d*),D	10 (2,2,2,1,3)	(IX + *d*) = D
DD 73 *d*	LD (IX+*d*),E	10 (2,2,2,1,3)	(IX + *d*) = E
DD 74 *d*	LD (IX+*d*),H	10 (2,2,2,1,3)	(IX + *d*) = H
DD 75 *d*	LD (IX+*d*),L	10 (2,2,2,1,3)	(IX + *d*) = L

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D
	•

Description

- **LD (IX+*d*),HL:** Loads the data in register L into the memory location whose address is the sum of the data in index register IX and a displacement value *d*. Then, loads the data in register H into the memory location whose address is the sum of the data in index register IX and a displacement value *d* plus 1.
- **LD (IX+*d*),*n*:** Loads the 8-bit constant *n* into the memory location whose address is the sum of index register IX and a displacement value *d*.
- **LD (IX+*d*),*r*:** Loads the data in register *r* (any of the registers A, B, C, D, E, H, or L) into the memory location whose address is the sum of the data in index register IX plus a displacement value *d*.

LD (IY+*d*),HL
LD (IY+*d*),*n*
LD (IY+*d*),*r*

N

Opcode	Instruction	Clocks	Operation
FD F4 *d*	LD (IY+*d*),HL	13 (2,2,2,1,3,3)	(IY + *d*) = L; (IY + *d* + 1) = H
FD 36 *d n*	LD (IY+*d*),*n*	11 (2,2,2,2,3)	(IY + *d*) = *n*
——	**LD (IY+*d*),*r***	**10 (2,2,2,1,3)**	**(IY + *d*) = *r***
FD 77 *d*	LD (IY+*d*),A	10 (2,2,2,1,3)	(IY + *d*) = A
FD 70 *d*	LD (IY+*d*),B	10 (2,2,2,1,3)	(IY + *d*) = B
FD 71 *d*	LD (IY+*d*),C	10 (2,2,2,1,3)	(IY + *d*) = C
FD 72 *d*	LD (IY+*d*),D	10 (2,2,2,1,3)	(IY + *d*) = D
FD 73 *d*	LD (IY+*d*),E	10 (2,2,2,1,3)	(IY + *d*) = E
FD 74 *d*	LD (IY+*d*),H	10 (2,2,2,1,3)	(IY + *d*) = H
FD 75 *d*	LD (IY+*d*),L	10 (2,2,2,1,3)	(IY + *d*) = L

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D
	•

Description

- **LD (IY+*d*),HL:** Loads the data in register L into the memory location whose address is the sum of the data in index register IY and a displacement value *d*. Then, loads the data in register H into the memory location whose address is the sum of the data in index register IY and a displacement value *d* plus 1.
- **LD (IY+*d*),*n*:** Loads the 8-bit constant *n* into the memory location whose address is the sum of the data in index register IY and a displacement value *d*.
- **LD (IY+*d*),*r*:** Loads the data in register *r* (any of the registers A, B, C, D, E, H, or L) into the memory location whose address is the sum of the data in index register IY plus a displacement value *d*.

LD (*mn*),A
LD (*mn*),HL
LD (*mn*),IX
LD (*mn*),IY
LD (*mn*),*ss*

Opcode	Instruction	Clocks	Operation
32 *n m*	LD (*mn*),A	a	(*mn*) = A
22 *n m*	LD (*mn*),HL	b	(*mn*) = L; (*mn* + 1) = H
DD 22 *n m*	LD (*mn*),IX	c	(*mn*) = $IX_{(low)}$; (*mn* + 1) = $IX_{(high)}$
FD 22 *n m*	LD (*mn*),IY	c	(*mn*) = $IY_{(low)}$; (*mn* + 1) = $IY_{(high)}$
——	**LD (*mn*),*ss***	**c**	**(*mn*) = $ss_{(low)}$; (*mn* + 1) = $ss_{(high)}$**
ED 43 *n m*	LD (*mn*),BC	c	(*mn*) = C; (*mn* + 1) = B
ED 53 *n m*	LD (*mn*),DE	c	(*mn*) = E; (*mn* + 1) = D
ED 63 *n m*	LD (*mn*),HL	c	(*mn*) = L; (*mn* + 1) = H
ED 73 *n m*	LD (*mn*),SP	c	(*mn*) = P; (*mn* + 1) = S
Clocking: (a)10 (2,2,2,1,3) (b)13 (2,2,2,1,3,3) (c)15 (2,2,2,2,1,3,3)			

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D
	•

Description

- **LD (*mn*),A:** Loads the memory location whose address is *mn* with the data in the Accumulator.
- **LD (*mn*),HL:** Loads the memory location whose address is *mn* with the data in register L, then loads the memory location whose address is 1 plus *mn* with the data in register H.
- **LD (*mn*),IX:** Loads the memory location whose address is *mn* with the low order byte of the data in index register IX, and the memory location whose address is 1 plus *mn* with the high order byte of the data in IX.
- **LD (*mn*),IY:** Loads the memory location whose address is *mn* with the low order byte of the data in index register IY, the memory location whose address is 1 plus *mn* with the high order byte of the data in IY into.
- **LD (*mn*),*ss*:** Loads the memory location whose address is *mn* with the low order byte of the data in word register *ss* (any of the word registers BC, DE, HL or SP). Then, loads the memory location whose address is 1 plus *mn* with the high order byte of the data in word register *ss*.

LD (SP+*n*),HL N
LD (SP+*n*),IX N
LD (SP+*n*),IY N

Opcode	Instruction	Clocks	Operation
D4 *n*	LD (SP+*n*),HL	11 (2,2,1,3,3)	(SP + *n*) = L; (SP + *n* + 1) = H
DD D4 *n*	LD (SP+*n*),IX	13 (2,2,2,1,3,3)	(SP + *n*) = $IX_{(low)}$; (SP + *n* + 1) = $IX_{(high)}$
FD D4 *n*	LP (SP+*n*),IY	13 (2,2,2,1,3,3)	(SP + *n*) = $IY_{(low)}$; (SP + *n* + 1) = $IY_{(high)}$

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

- **LD (SP+*n*),HL:** Loads the data in the register L into the memory location whose address is the sum of the data in the Stack Pointer, SP, and the displacement *n*. Then loads the data in the register H into the memory location whose address is the sum of the data in SP, the displacement *n*, and 1.
- **LD (SP+*n*),IX:** Loads the low order byte of the data in index register IX into the memory location whose address is the sum of the data in the Stack Pointer, SP, and the displacement *n*. Then loads the high order byte of the data in IX into the memory location whose address is the sum of data in SP, the displacement *n*, and 1.
- **LP (SP+*n*),IY:** Loads the low order byte of the data in index register IY into the memory location whose address is the sum of the data in the Stack Pointer, SP, and the displacement *n*. Then loads the high order byte of the data in IY into the memory location whose address is the sum of data in SP, the displacement *n*, and 1.

LD A,(BC)
LD A,(DE)
LD A,(*mn*)

Opcode	Instruction	Clocks	Operation
0A	LD A,(BC)	6 (2,2,2)	A = (BC)
1A	LD A,(DE)	6 (2,2,2)	A = (DE)
3A *n m*	LD A,(*mn*)	9 (2,2,2,1,2)	A = (*mn*)

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP
	•	

I/O	
S	D
•	

Description

Loads the Accumulator with the data whose address in memory is:

- the data in word register BC, or
- the data in word register DE, or
- the 16-bit constant *mn*.

LD A,EIR
LD A,IIR

Opcode	Instruction	Clocks	Operation
ED 57	LD A,EIR	4 (2,2)	A = EIR
ED 5F	LD A,IIR	4 (2,2)	A = IIR

Flags							
S	Z				L/V		C
•	•				-		-

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

- **LD A,EIR:** Loads the Accumulator with the data in the External Interrupt Register, EIR. The EIR is used to specify the Most Significant Byte (MSB) of the External Interrupt address. The value loaded in the EIR is concatenated with the apprporiate External Interrupt address to form the 16-bit ISR starting address.
- **LD A,IIR:** Loads the Accumulator with the data in the Internal Interrupt Register, IIR. The IIR is used to specify the Most Significant Byte (MSB) of the Internal Peripheral Interrupt address. The value loaded in the IIR is concatenated with the apprporiate Internal Peripheral address to form the 16-bit ISR starting address for that peripheral.

LD A,XPC NP

Opcode	Instruction	Clocks	Operation
ED 77	LD A,XPC	4 (2,2)	A = XPC

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP
	•	

I/O	
S	D

Description

Loads the Accumulator with the data in the Extension of the Program Counter, XPC.

LD *dd*,(*mn*)

Opcode	Instruction	Clocks	Operation
—	**LD *dd*,(*mn*)**	**13 (2,2,2,2,1,2,2)**	***dd*$_{(low)}$ = (*mn*);** ***dd*$_{(high)}$ = (*mn* + 1)**
ED 4B *n m*	LD BC,(*mn*)	13 (2,2,2,2,1,2,2)	C = (*mn*); B = (*mn* + 1)
ED 5B *n m*	LD DE,(*mn*)	13 (2,2,2,2,1,2,2)	E = (*mn*); D = (*mn* + 1)
ED 6B *n m*	LD IX,(*mn*)	13 (2,2,2,2,1,2,2)	X = (*mn*); I = (*mn* + 1)
ED 7B *n m*	LD SP,(*mn*)	13 (2,2,2,2,1,2,2)	P = (*mn*); S = (*mn* + 1)

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP
	•	

I/O	
S	D
•	

Description

Loads the low order byte of the word register *dd* (any of the word registers BC, DE, IX, or SP) with the data at memory address *mn*. Then loads the high order byte of the word register *dd* with data at memory address *mn* plus 1.

LD *dd'*,BC N
LD *dd'*,DE N

Opcode	Instruction	Clocks	Operation
—	**LD *dd'*,BC**	**4 (2,2)**	***dd'* = BC**
ED 49	LD BC',BC	4 (2,2)	BC' = BC
ED 59	LD DE',BC	4 (2,2)	DE' = BC
ED 69	LD HL',BC	4 (2,2)	HL' = BC
—	**LD *dd'*,DE**	**4 (2,2)**	***dd'* = DE**
ED 41	LD BC',DE	4 (2,2)	BC' = DE
ED 51	LD DE',DE	4 (2,2)	DE' = DE
ED 61	LD HL',DE	4 (2,2)	HL' = DE

Flags									ALTD				I/O	
S	Z				L/V		C		F	R	SP		S	D
-	-				-		-							

Description

Loads the alternate register pair *dd'* (any of the registers BC', DE', or HL') with the data in the register pair BC or the register pair DE.

LD *dd,mn*

Opcode	Instruction	Clocks	Operation
——	**LD *dd,mn***	**6 (2,2,2)**	***dd* = *mn***
01 *n m*	LD BC,*mn*	6 (2,2,2)	BC = *mn*
11 *n m*	LD DE,*mn*	6 (2,2,2)	DE = *mn*
21 *n m*	LD IX,*mn*	6 (2,2,2)	IX = *mn*
31 *n m*	LD SP,*mn*	6 (2,2,2)	SP = *mn*

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP
	•	

I/O	
S	D

Description

Loads the register pair *dd* (any of the register pairs BC, DE, IX, or SP) with the 16-bit value *mn*.

LD EIR,A
LD IIR,A

Opcode	Instruction	Clocks	Operation
ED 47	LD EIR,A	4 (2,2)	EIR = A
ED 4F	LD IIR,A	4 (2,2)	IIR = A

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

- **LD EIR,A:** Loads the External Interrupt Register, EIR, with the data in the Accumulator. The EIR is used to specify the Most Significant Byte (MSB) of the External Interrupt address. The value loaded in the EIR is concatenated with the apprporiate External Interrupt address to form the 16-bit ISR starting address.
- **LD IIR,A:** Loads the Internal Interrupt Register, IIR, with the data in the Accumulator. The IIR is used to specify the Most Significant Byte (MSB) of the Internal Peripheral Interrupt address. The value loaded in the IIR is concatenated with the apprporiate Internal Peripheral address to form the 16-bit ISR starting address for that peripheral.

LD HL,(*mn*)LD HL,(HL+*d*) N
LD HL,(IX+*d*) N
LD HL,(IY+*d*) N

Opcode	Instruction	Clocks	Operation
2A *mn*	LD HL,(*mn*)	11 (2,2,2,1,2,2)	L = (*mn*); H = (*mn* + 1)
DD E4 *d*	LD HL,(HL+*d*)	11 (2,2,2,1,2,2)	L = (HL + *d*); H = (HL + *d* + 1)
E4 *d*	LD HL,(IX+*d*)	9 (2,2,1,2,2)	L = (IX + *d*); H = (IX + *d* + 1)
FD E4 *d*	LD HL,(IY+*d*)	11 (2,2,2,1,2,2)	L = (IY + *d*); H = (IY + *d* + 1)

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP
	•	

I/O	
S	D
•	

Description

- **LD HL,(*mn*):** Loads the register L with the data whose address is *mn* and loads the register H with the data whose address is *mn* plus 1.
- **LD HL,(HL+*d*):** Loads the register L with the data whose address is the data in word register HL plus a displacement *d*. Then loads the register H with the data whose address is the data in word register HL plus a displacement *d* plus 1.
- **LD HL,(IX+*d*):** Loads the register L with the data whose address is the data in index register IX plus a displacement *d*. Then loads the register H with the data whose address is the data in index register IX plus a displacement *d* plus 1.
- **LD HL,(IY+*d*):** Loads the register L with the data whose address is the data in index register IY plus a displacement *d*. Then loads the register H with the data whose address is the data in index register IY plus a displacement *d* plus 1.

LD HL,(SP+*n*) N

Opcode	Instruction	Clocks	Operation
C4 *n*	LD HL,(SP+*n*)	9 (2,2,1,2,2)	L = (SP + *n*); H = (SP + *n* + 1)

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP
	•	

I/O	
S	D

Description

Loads the register L with the data whose address is the data in index register SP plus a displacement *d*. Then loads the register H with the data whose address is the data in index register SP plus a displacement *d* plus 1.

LD HL,IX N
LD HL,IY N

Opcode	Instruction	Clocks	Operation
DD 7C	LD HL,IX	4 (2,2)	HL = IX
FD 7C	LD HL,IY	4 (2,2)	HL = IY

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP
	•	

I/O	
S	D

Description

- **LD HL,IX:** Loads the word register HL with the data in index register IX.
- **LD HL,IY:** Loads the word register HL with the data in index register IY.

LD IX,(*mn*)

Opcode	Instruction	Clocks	Operation
DD 2A *n m*	LD IX,(*mn*)	13*	$IX_{(low)}$ = (*mn*); $IX_{(high)}$ = (*mn* + 1)
*Clocking: 13 (2,2,2,2,1,2,2)			

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D
•	

Description

Loads the low order byte of index register IX with the data whose address is *mn*. Then loads the high order byte of IX with the data whose address is *mn* plus 1.

LD IX,(SP+*n*) N

Opcode	Instruction	Clocks	Operation
DD C4 *n*	LD IX,(SP+*n*)	11*	$IX_{(low)}$ = (SP + *n*); $IX_{(high)}$ = (SP + *n* + 1)
*Clocking: 11 (2,2,2,1,2,2)			

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

Loads the low order byte of index register IX with the data whose address is the data in the Stack Pointer, SP, plus a displacement *n*. Then loads the high order byte of IX with the data whose address is the data in the Stack Pointer register plus a displacement *n* plus 1.

LD IX,HL
LD IX,*mn*
LD IY,HL
LD IY,*mn*

Opcode	Instruction	Clocks	Operation
DD 7D	LD IX,HL	4 (2,2)	IX = HL
DD 21 *n m*	LD IX,*mn*	8 (2,2,2,2)	IX = *mn*
FD 7D	LD IY,HL	4 (2,2)	IY = HL
FD 21 *n m*	LD IY,*mn*	8 (2,2,2,2)	IY = *mn*

Flags				ALTD			I/O	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

- **LD IX,HL:** Loads the index register IX with the data in word register HL.
- **LD IX,*mn*:** Loads the index register IX with the 16-bit constant *mn*.
- **LD IY,HL:** Loads the index register IY with the data in word register HL.
- **LD IX,*mn*:** Loads the index register IY with the 16-bit constant *mn*.

Description

Loads the low order byte of index register IY with the data at the address *mn* and loads the high order byte of IY with the data at the address *mn*+1.

LD IY,(SP+*n*) N

Opcode	In,5ruction	Clocks	Operation
FD C4 *n*	LD IY,(SP+*n*)	11*	$IY_{(low)}$ = (SP + *n*); $IY_{(high)}$ = (SP + *n* + 1)
*Clocking: 11 (2,2,2,1,2,2)			

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

Loads the low order byte of index register IY with the data whose address is the data in the Stack Pointer register SP plus a displacement *n*. Then loads the high order byte of IY with the data whose address is the data in the Stack Pointer register plus a displacement *n* plus 1.

LD *r*,(HL)
LD *r*,(IX+*d*)
LD *r*,(IY+*d*)

Opcode	Instruction	Clocks	Operation
——	**LD *r*,(HL)**	**5 (2,1,2)**	***r* = (HL)**
7E	LD A,(HL)	5 (2,1,2)	A = (HL)
46	LD B,(HL)	5 (2,1,2)	B = (HL)
4E	LD C,(HL)	5 (2,1,2)	C = (HL)
56	LD D,(HL)	5 (2,1,2)	D = (HL)
5E	LD E,(HL)	5 (2,1,2)	E = (HL)
66	LD H,(HL)	5 (2,1,2)	H = (HL)
6E	LD L,(HL)	5 (2,1,2)	L = (HL)
——	**LD *r*,(IX+*d*)**	**9 (2,2,2,1,2)**	***r* = (IX + *d*)**
DD 7E *d*	LD A,(IX+*d*)	9 (2,2,2,1,2)	A = (IX + *d*)
DD 46 *d*	LD B,(IX+*d*)	9 (2,2,2,1,2)	B = (IX + *d*)
DD 4E *d*	LD C,(IX+*d*)	9 (2,2,2,1,2)	C = (IX + *d*)
DD 56 *d*	LD D,(IX+*d*)	9 (2,2,2,1,2)	D = (IX + *d*)
DD 5E *d*	LD E,(IX+*d*)	9 (2,2,2,1,2)	E = (IX + *d*)
DD 66 *d*	LD H,(IX+*d*)	9 (2,2,2,1,2)	H = (IX + *d*)
DD 6E *d*	LD L,(IX+*d*)	9 (2,2,2,1,2)	L = (IX + *d*)
——	**LD *r*,(IY+*d*)**	**9 (2,2,2,1,2)**	***r* = (IY + *d*)**
FD 7E *d*	LD A,(IY+*d*)	9 (2,2,2,1,2)	A = (IY + *d*)
FD 46 *d*	LD B,(IY+*d*)	9 (2,2,2,1,2)	B = (IY + *d*)
FD 4E *d*	LD C,(IY+*d*)	9 (2,2,2,1,2)	C = (IY + *d*)
FD 56 *d*	LD D,(IY+*d*)	9 (2,2,2,1,2)	D = (IY + *d*)
FD 5E *d*	LD E,(IY+*d*)	9 (2,2,2,1,2)	E = (IY + *d*)
FD 66 *d*	LD H,(IY+*d*)	9 (2,2,2,1,2)	H = (IY + *d*)
FD 6E *d*	LD L,(IY+*d*)	9 (2,2,2,1,2)	L = (IY + *d*)

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP
	•	

I/O	
S	D
•	

Description

Loads the register *r* (any of the registers A, B, C, D, E, H, or L) with the data whose address is:

- the data in word register HL, or
- the sum of the data in index register IX and a displacement *d*, or
- the sum of the data in index register IY and a displacement *d*.

LD *r,n*

Opcode	Instruction	Clocks	Operation
——	**LD *r,n***	**4 (2,2)**	***r* = *n***
3E *n*	LD A,*n*	4 (2,2)	A = *n*
06 *n*	LD B,*n*	4 (2,2)	B = *n*
0E *n*	LD C,*n*	4 (2,2)	C = *n*
16 *n*	LD D,*n*	4 (2,2)	D = *n*
1E *n*	LD E,*n*	4 (2,2)	E = *n*
26 *n*	LD H,*n*	4 (2,2)	H = *n*
2E *n*	LD L,*n*	4 (2,2)	L = *n*

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP
	•	

I/O	
S	D

Description

Loads the register *r* (any of the registers A, B, C, D, E, H, or L) with the 8-bit constant *n*.

LD *r,g*

Opcode								Instruction	Clocks	Operation
								LD *r,g*	2	*r* = *r'*
r,g	A'	B'	C'	D'	E'	H'	L'			
A	7F	78	79	7A	7B	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	6B	6C	6D			

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP
	•	

I/O	
S	D

Description

Loads the one-byte register *r* (any of the registers A, B, C, D, E, H, or L) with the data in another one-byte register *g* (any of the registers A, B, C, D, E, H, or L).

LD SP,HL P
LD SP,IX P
LD SP,IY P

Opcode	Instruction	Clocks	Operation
F9	LD SP,HL	2	SP = HL
DD F9	LD SP,IX	4 (2,2)	SP = IX
FD F9	LD SP,IY	4 (2,2)	SP = IY

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

Loads the Stack Pointer register, SP, with the data in (a) the word register HL, (b) the data in index register IX, or (c) the data in index register IY.

LD XPC,A NP

Opcode	Instruction	Clocks	Operation
ED 67	LD XPC,A	4 (2,2)	XPC = A

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

Loads the Extension of the Program Counter, XPC, with the data in the Accumulator.

LDD
LDDR
LDI
LDIR

Opcode	Instruction	Clocks	Operation
ED A8	LDD	10 (2,2,1,2,3)	(DE) = (HL); BC = BC - 1; DE = DE - 1; HL = HL - 1
ED B8	LDDR	6 + 7i (2,2,1,(2,3,2)i,1)	While {BC != 0} repeat: (DE) = (HL); BC = BC - 1; DE = DE - 1; HL = HL - 1
ED A0	LDI	10 (2,2,1,2,3)	(DE) = (HL); BC = BC - 1; DE = DE + 1; HL = HL + 1
ED B0	LDIR	6 + 7i (2,2,1,(2,3,2)i,1)	While {BC != 0} repeat: (DE) = (HL); BC = BC - 1; DE = DE + 1; HL = HL + 1

Flags							
S	Z				L/V		C
-	-				•		-

ALTD		
F	R	SP

I/O	
S	D
	•

Description

- **LDD:** Loads the memory location whose address is in word register DE with the data at the address in word register HL. Then it decrements the data in word registers BC, DE, and HL.
- **LDDR:** While the data in the register pair BC does not equal 0 then the memory location whose address is in word register DE is loaded with the data at the address in word register HL. Then it decrements the data in word registers BC, DE, and HL. The instruction then repeats until BC equals zero.
- **LDI:** Loads the memory location whose address is in word register DE with the data at the address in word register HL. Then the data in word register BC is decremented and the data in word registers DE and HL is incremented.
- **LDIR:** While the data in the register pair BC does not equal 0 then the memory location whose address is in word register DE is loaded with the data at the address in word register HL. Then the data in word register BC is decremented and the data in word registers DE and HL are incremented. The instruction then repeats until BC equals zero.

LDP (HL),HL N
LDP (IX),HL N
LDP (IY),HL N

Opcode	Instruction	Clocks	Operation
ED 64	LDP (HL),HL	12 (2,2,2,3,3)	(HL) = L; (HL + 1) = H. (Addr[19:16] = A[3:0])
DD 64	LDP (IX),HL	12 (2,2,2,3,3)	(IX) = L; (IX + 1) = H. (Addr[19:16] = A[3:0])
FD 64	LDP (IY),HL	12 (2,2,2,3,3)	(IY) = L; (IY + 1) = H. (Addr[19:16] = A[3:0])

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

These instructions are used to access 20-bit addresses. In all cases, the four most significant bits of the 20-bit address (bits 19 through 16) are defined as the four least significant bits of the Accumulator (bits 3 though 0). The LDP instructions bypass the MMU's address translation unit for direct access to the 20-bit memory address space.

- **LDP HL,(HL):** Loads the memory location whose 16 least significant bits of its 20-bit address are the data in paired regsiter HL with the data in the register L, and then loads the following 20-bit address with the data in the register H.
- **LDP HL,(IX):** Loads the memory location whose 16 least significant bits of its 20-bit address are the data in index regsiter IX with the data in the register L, and then loads the following 20-bit address with the data in the register H.
- **LDP HL,(IY):** Loads the memory location whose 16 least significant bits of its 20-bit address are the data in index regsiter IY with the data in the register L, and then loads the following 20-bit address with the data in the register H.

LDP (*mn*),HL N
LDP (*mn*),IX N
LDP (*mn*),IY N

Opcode	Instruction	Clocks	Operation
ED 65 *n m*	LDP (*mn*),HL	15*	(*mn*) = L; (*mn* + 1) = H. (Addr[19:16] = A[3:0])
DD 65 *n m*	LDP (*mn*),IX	15*	(*mn*) = $IX_{(low)}$; (*mn* + 1) = $IX_{(high)}$. (Addr[19:16] = A[3:0])
FD 65 *n m*	LDP (*mn*),IY	15*	(*mn*) = $IY_{(low)}$; (*mn* + 1) = $IY_{(high)}$. (Addr[19:16] = A[3:0])
*Clocking: 15 (2,2,2,2,1,3,3)			

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

These instructions are used to access 20-bit addresses. In all cases, the four most significant bits of the 20-bit address (bits 19 through 16) are defined as the four least significant bits of the Accumulator (bits 3 though 0). The LDP instructions bypass the MMU's address translation unit for direct access to the 20-bit memory address space.

- **LDP (*mn*),HL:** Loads the memory location whose 16 least significant bits of its 20-bit address are the 16-bit constant *mn* with the data in the register L, and then loads the following memory location with the data in the register H.
- **LDP (*mn*),IX:** Loads the memory location whose 16 least significant bits of its 20-bit address are the 16-bit constant *mn* with the the low order byte of index register IX, and then loads the following memory location with the high order byte of index register IX.
- **LDP (*mn*),IY:** Loads the memory location whose 16 least significant bits of its 20-bit address are the 16-bit constant *mn* with the the low order byte of index register IY, and then loads the following memory location with the high order byte of index register IY.

LDP HL,(HL) N
LDP HL,(IX) N
LDP HL,(IY) N

Opcode	Instruction	Clocks	Operation
ED 6C	LDP HL,(HL)	10 (2,2,2,2,2)	L = (HL); H = (HL + 1). (Addr[19:16] = A[3:0])
DD 6C	LDP HL,(IX)	10 (2,2,2,2,2)	L = (IX); H = (IX + 1). (Addr[19:16] = A[3:0])
FD 6C	LDP HL,(IY)	10 (2,2,2,2,2)	L = (IY); H = (IY + 1). (Addr[19:16] = A[3:0])

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

These instructions are used to access 20-bit addresses. In all cases, the four most significant bits of the 20-bit address (bits 19 through 16) are defined as the four least significant bits of the Accumulator (bits 3 though 0). The LDP instructions bypass the MMU's address translation unit for direct access to the 20-bit memory address space.

- **LDP HL,(HL):** Loads the register L with the data whose 16 least significant bits of its 20-bit address are the data in paired regsiter HL, and then loads the register H with the data in the following 20-bit address.
- **LDP HL,(IX):** Loads the register L with the data whose 16 least significant bits of its 20-bit address are the data in index regsiter IX, and then loads the register H with the data in the following 20-bit address.
- **LDP HL,(IY):** Loads the register L with the data whose 16 least significant bits of its 20-bit address are the data in index regsiter IY, and then loads the register H with the data in the following 20-bit address.

LDP HL,(*mn*) N
LDP IX,(*mn*) N
LDP IY,(*mn*) N

Opcode	Instruction	Clocks	Operation
ED 6D *n m*	LDP HL,(*mn*)	13*	L = (*mn*); H = (*mn* + 1). (Addr[19:16] = A[3:0])
DD 6D *n m*	LDP IX,(*mn*)	13*	$IX_{(low)}$ = (*mn*); $IX_{(high)}$ = (*mn* + 1). (Addr[19:16] = A[3:0])
FD 6D *n m*	LDP IY,(*mn*)	13*	$IY_{(low)}$ = (*mn*); $IY_{(high)}$ = (*mn* + 1). (Addr[19:16] = A[3:0])
*Clocking: 13 (2,2,2,2,1,2,2)			

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

These instructions are used to access 20-bit addresses. In all cases, the four most significant bits of the 20-bit address (bits 19 through 16) are defined as the four least significant bits of the Accumulator (bits 3 though 0). The LDP instructions bypass the MMU's address translation unit for direct access to the 20-bit memory address space.

- **LDP HL,(*mn*):** Loads the register L with the data whose 16 least significant bits of its 20-bit address are the 16-bit constant *mn*, and then loads the register H with the data in the following 20-bit address.
- **LDP IX,(*mn*):** Loads the low order byte of index register IX with the data whose 16 least significant bits of its 20-bit address are the 16-bit constant *mn*, and then loads the high order byte of IX with the data in the following 20-bit address.
- **LDP IY,(*mn*):** Loads the low order byte of index register IY with the data whose 16 least significant bits of its 20-bit address are the 16-bit constant *mn*, and then loads the high order byte of IY with the data in the following 20-bit address.

LJP *x,mn* N

Opcode	Instruction	Clocks	Operation
C7 *n m x*	LJP *x,mn*	10 (2,2,2,2,2)	XPC = *x*; PC = *mn*

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

This instruction is similar to the JP mn instruction in that it transfers program execution to the memory location specified by the 16-bit address, mn. LJP is special in that it allows a jump to be made to a computed address in XMEM. Note that the value of XPC and consequently the address space defined by the XPC is dynamically changed with the LJP instructions.

The instruction loads the XPC, with the 8-bit constant *x*. Then loads the Program Counter, PC, with the 16-bit constant *mn*, which must be in the range E000–FFFF.

This instruction recognizes labels when used in the Dynamic C assembler.

LRET N

Opcode	Instruction	Clocks	Operation
ED 45	LRET	13 (2,2,1,2,2,2,2)	$PC_{(low)}$ = (SP); $PC_{(high)}$ = (SP+1); XPC = (SP + 2); SP = SP + 3

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

The LRET transfers execution from a subroutine to the calling program by popping the Program Counter and the XPC off of the Stack, in order to return from a LCALL operation.

The instruction first loads the low order byte of the Program Counter with the data whose address is the data in the Stack Pointer, SP. Then it loads the high order byte of the PC with the data in whose address is the sum data in the SP and 1. Then it loads the Extension of the Program Counter, the XPC, with the data whose address is the data in the SP plus 2. Finally it adds three to the value in the SP and stores the result in the SP.

MUL N

Opcode	Instruction	Clocks	Operation
F7	MUL	12 (2,10)	HL:BC = BC • DE

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

A multiplication operation is performed on the contents of the 16-bit binary integers contained in the BC and DE registers. The signed 32-bit result is placed in the HL (bits 31 through 16) and BC (bits 15 through 0) registers.

If the multiplier is negative, the hardware takes its 2's-complement as multiplication is being performed. If the multiplier is positive, it is passed unchanged. If there is a carry from this stage of the 2's complement operation, it is passed it to the next stage.

Examples:

```
LD BC, 0FFFFh    ;BC gets -1
LD DE, 0FFFFh    ;DE gets -1
MUL              ;HL|BC = 1, HL gets 0000h, BC gets 0001h
```

In the above example, the 2's complement of FFFFh is 0001h.

```
LD BC, 0FFFFh    ;BC gets -1
LD DE, 00001h    ;DE gets 1
MUL              ;HL|BC = -1, HL gets FFFFh, BC gets FFFFh
```

NEG

Opcode	Instruction	Clocks	Operation
ED 44	NEG	4 (2,2)	A = 0 - A

Flags							
S	Z				L/V		C
•	•				V		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Subtracts the value of the data in the Accumulator from zero and stores the result in the Accumulator.

NOP

Opcode	Instruction	Clocks	Operation
00	NOP	2	No operation

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

No operation is performed during this cycle.

OR (HL)
OR (IX+*d*)
OR (IY+*d*)

Opcode	Instruction	Clocks	Operation
B6	OR (HL)	5 (2,1,2)	A = A | (HL)
DD B6 *d*	OR (IX+*d*)	9 (2,2,2,1,2)	A = A | (IX+*d*)
FD B6 *d*	OR (IY+*d*)	9 (2,2,2,1,2)	A = A | (IY+*d*)

Flags							
S	Z				L/V		C
•	•				L		0

ALTD		
F	R	SP
•	•	

I/O	
S	D
•	

Description

Performs a logical OR operation between the byte in the Accumulator and the byte whose address is (a) in the word register HL, (b) the sum of the data in index register IX and a displacement *d*, or (c) the sum of the data in index register IY and a displacement *d*.

The relative bits of each byte are compared (i.e., the bit 1 of both bytes are compared, the bit 2 of both bytes are compared, etc.) and the associated bit in the result byte is set if either of the compared bits is set. The result is stored in the Accumulator.

Example

If the byte in the Accumulator is 0100 1100 and the byte in the memory location pointed to by HL is 1110 0101, the operation:

```
OR (HL)
```

would result in the Accumulator containing 1110 1101.

OR HL,DE N

Opcode	Instruction	Clocks	Operation
EC	OR HL,DE	2	HL = HL | DE

Flags							
S	Z				L/V		C
•	•				L		0

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Performs a logical OR between the data in word register HL and the data in word register DE. The relative bits of each byte are compared (i.e., the bit 1 of both bytes are compared, the bit 2 of both bytes are compared, etc.) and the associated bit in the result byte is set if either of the compared bits is set. The result is stored in HL.

OR IX,DE N
OR IY,DE N

Opcode	Instruction	Clocks	Operation
DD EC	OR IX,DE	4 (2,2)	IX = IX | DE
FD EC	OR IY,DE	4 (2,2)	IY = IY | DE

Flags							
S	Z				L/V		C
•	•				L		0

ALTD		
F	R	SP
•		

I/O	
S	D

Description

- **OR IX,DE:** Performs a logical OR operation between the data in index register IX and the data in word registers DE. The result is stored in IX
- **OR IY,DE:** Performs a logical OR operation between the data in index register IY and the data in word register DE. The result is stored in IY

The relative bits of each byte are compared (i.e., the bit 1 of both bytes are compared, the bit 2 of both bytes are compared, etc.) and the associated bit in the result byte is set if either of the compared bits is set.

OR *n*
OR *r*

Opcode	Instruction	Clocks	Operation
F6 *n*	OR *n*	4 (2,2)	A = A | *n*
——	**OR *r***	**2**	**A = A | *r***
B7	OR A	2	A = A | A
B0	OR B	2	A = A | B
B1	OR C	2	A = A | C
B2	OR D	2	A = A | D
B3	OR E	2	A = A | E
B4	OR H	2	A = A | H
B5	OR L	2	A = A | L

Flags							
S	Z				L/V		C
•	•				L		0

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

- **OR *n*:** Performs a logical OR operation between the byte in the Accumulator and the 8-bit constant *n*.
- **OR *r*:** Performs a logical OR operation between the byte in the Accumulator an the byte in register *r* (any of the registers A, B, C, D, E, H, or L).

The relative bits of each byte are compared (i.e., the bit 1 of both bytes are compared, the bit 2 of both bytes are compared, etc.) and the associated bit in the result byte is set if either of the compared bits is set. The result is stored in the Accumulator.

POP IP
POP IX
POP IY

NP

Opcode	Instruction	Clocks	Operation
ED 7E	POP IP	7 (2,2,1,2)	IP = (SP); SP = SP + 1
DD E1	POP IX	9 (2,2,1,2,2)	$IX_{(low)}$ = (SP); $IX_{(high)}$ = (SP + 1); SP = SP + 2
FD E1	POP IY	9 (2,2,1,2,2)	$IY_{(low)}$ = (SP); $IY_{(high)}$ = (SP + 1); SP = SP + 2

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

- **POP IP:** Loads the Interrupt Priority Register, IP, with the data at the memory location in the Stack Pointer, SP, and then increments the data in SP.
- **POP IX:** Loads the low order byte of index register IX with the data at the memory address in the Stack Pointer, SP, then loads the high order byte of IX with the data at the address immediately following the one held in SP. SP is then incremented twice.
- **POP IY:** Loads the low order byte of index register IY with the data at the memory address in the Stack Pointer, SP, then loads the high order byte of IY with the data at the memory address immediately following the one held in SP. SP is then incremented twice.

POP *zz*

Opcode	Instruction	Clocks	Operation
——	**POP *zz***	**7 (2,1,2,2)**	***zz*$_{(low)}$ = (SP); *zz*$_{(high)}$ = (SP + 1); SP = SP + 2**
F1	POP AF	7 (2,1,2,2)	F = (SP); A = (SP + 1); SP = SP + 2
C1	POP BC	7 (2,1,2,2)	C = (SP); B = (SP + 1); SP = SP + 2
D1	POP DE	7 (2,1,2,2)	E = (SP); D = (SP + 1); SP = SP + 2
E1	POP HL	7 (2,1,2,2)	L = (SP); H = (SP + 1); SP = SP + 2

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP
	•	

I/O	
S	D

Description

Loads the low order byte of the word register *zz* (any of the word registers AF, BC, DE, or HL) with the data at the memory address in the Stack Pointer, SP, then loads the high order byte of *zz* with the data at the memory address immediately following the one held in SP. SP is then incremented twice.

PUSH IP
PUSH IX
PUSH IY

N

Opcode	Instruction	Clocks	Operation
ED 76	PUSH IP	9 (2,2,2,3)	(SP - 1) = IP; SP = SP - 1
DD E5	PUSH IX	12 (2,2,2,3,3)	(SP - 1) = $IX_{(high)}$; (SP - 2) = $IX_{(low)}$; SP = SP - 2
FD E5	PUSH IY	12 (2,2,2,3,3)	(SP - 1) = $IY_{(high)}$; (SP - 2) = $IY_{(low)}$; SP = SP - 2

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

- **PUSH IP:** Loads the location in memory whose address is 1 less that the data held in the Stack Pointer, SP, with the data in the Interrupt Priority Register IP. Then decrements SP.
- **PUSH IX:** Loads the memory location with the address 1 less than the data in the Stack Pointer, SP, with the high order byte of the data in index register IX, and loads the memory location with the address two less than the data in SP with the low order byte of the data in IX. Then SP is decremented twice.
- **PUSH IY:** Loads the memory location with the address 1 less than the data in the Stack Pointer, SP, with the high order byte of the data in index register IX, and loads the memory location with the address two less than the data in SP with the low order byte of the data in IX. Then SP is decremented twice.

PUSH *zz*

Opcode	Instruction	Clocks	Operation
——	**PUSH *zz***	**10 (2,2,3,3)**	**(SP - 1) = zz$_{(high)}$; (SP - 2) = zz$_{(low)}$; SP = SP - 2**
F5	PUSH AF	10 (2,2,3,3)	(SP - 1) = A; (SP - 2) = F; SP = SP - 2
C5	PUSH BC	10 (2,2,3,3)	(SP - 1) = B; (SP - 2) = C; SP = SP - 2
D5	PUSH DE	10 (2,2,3,3)	(SP - 1) = D; (SP - 2) = E; SP = SP - 2
E5	PUSH HL	10 (2,2,3,3)	(SP - 1) = H; (SP - 2) = L; SP = SP - 2

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

Loads the memory location with the address 1 less than the data in the Stack Pointer, SP, with the high order byte of the data in word register *zz* (any of the word register AF, BC, DE, or HL), and loads the memory location with the address two less than the data in SP with the low order byte of the data in *zz*. Then SP is decremented twice.

RES *b*,(HL)
RES *b*,(IX+*d*)
RES *b*,(IY+*d*)

Opcode	Instruction	Clocks	Operation
——	**RES *b*,(HL)**	**10***	**(HL) = (HL) & ~bit *b***
CB 86	RES bit 0,(HL)	10*	(HL) = (HL) & ~bit 0
CB 8E	RES bit 1,(HL)	10*	(HL) = (HL) & ~bit 1
CB 96	RES bit 2,(HL)	10*	(HL) = (HL) & ~bit 2
CB 9E	RES bit 3,(HL)	10*	(HL) = (HL) & ~bit 3
CB A6	RES bit 4,(HL)	10*	(HL) = (HL) & ~bit 4
CB AE	RES bit 5,(HL)	10*	(HL) = (HL) & ~bit 5
CB B6	RES bit 6,(HL)	10*	(HL) = (HL) & ~bit 6
CB BE	RES bit 7,(HL)	10*	(HL) = (HL) & ~bit 7
——	**RES *b*,(IX+*d*)**	**13****	**(IX + *d*) = (IX + *d*) & ~bit**
DD CB *d* 86	RES bit 0,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) & ~bit 0
DD CB *d* 8E	RES bit 1,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) & ~bit 1
DD CB *d* 96	RES bit 2,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) & ~bit 2
DD CB *d* 9E	RES bit 3,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) & ~bit 3
DD CB *d* A6	RES bit 4,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) & ~bit 4
DD CB *d* AE	RES bit 5,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) & ~bit 5
DD CB *d* B6	RES bit 6,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) & ~bit 6
DD CB *d* BE	RES bit 7,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) & ~bit 7
——	**RES *b*,(IY+*d*)**	**13****	**(IY + *d*) = (IY + *d*) & ~bit**
FD CB *d* 86	RES bit 0,(IY+*d*)	13**	(IY + *d*) = (IY + *d*) & ~bit 0
FD CB *d* 8E	RES bit 1,(IY+*d*)	13**	(IY + *d*) = (IY + *d*) & ~bit 1
FD CB *d* 96	RES bit 2,(IY+*d*)	13**	(IY + *d*) = (IY + *d*) & ~bit 2
FD CB *d* 9E	RES bit 3,(IY+*d*)	13**	(IY + *d*) = (IY + *d*) & ~bit 3
FD CB *d* A6	RES bit 4,(IY+*d*)	13**	(IY + *d*) = (IY + *d*) & ~bit 4
FD CB *d* AE	RES bit 5,(IY+*d*)	13**	(IY + *d*) = (IY + *d*) & ~bit 5
FD CB *d* B6	RES bit 6,(IY+*d*)	13**	(IY + *d*) = (IY + *d*) & ~bit 6
FD CB *d* BE	RES bit 7,(IY+*d*)	13**	(IY + *d*) = (IY + *d*) & ~bit 7
Clocking: *10 (2,2,1,2,3) **13 (2,2,2,2,2,3)			

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D
	•

Description

Resets bit *b* (any of the bits 0, 1, 2, 3, 4, 5, 6, or 7) of the data whose address is:

- held in word register HL, or
- the sum of the data in index register IX and a displacement *d*, or
- the sum of the data in index register IX and a displacement *d*.

The bit is reset by performing a logical AND between the selected bit and its complement.

RES *b,r*

Opcode	Instruction	Clocks	Operation
	RES *b,r*	4 (2,2)	*r* = *r* & ~bit

b,r	A	B	C	D	E	H	L
CB(0)	87	80	81	82	83	84	85
CB(1)	8F	88	89	8A	8B	8C	8D
CB(2)	97	90	91	92	93	94	95
CB(3)	9F	98	99	9A	9B	9C	9D
CB(4)	A7	A0	A1	A2	A3	A4	A5
CB(5)	AF	A8	A9	AA	AB	AC	AD
CB(6)	B7	B0	B1	B2	B3	B4	B5
CB(7)	BF	B8	B9	BA	BB	BC	BD

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP
	•	

I/O	
S	D

Description

Resets bit *b* (any of the bits 0, 1, 2, 3, 4, 5, 6, or 7) of the data whose address is held in the register *r* (any of the register A, B, C, D, E, H, or L).

The bit is reset by performing a logical AND between the selected bit and its complement.

RET

Opcode	Instruction	Clocks	Operation
C9	RET	8 (2,1,2,2,1)	$PC_{(low)}$ = (SP); $PC_{(high)}$ = (SP + 1); SP = SP + 2

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

RET transefers execution from a subroutine to the program that called it. First it loads the low order byte of the Program Counter, PC, with the data at the memory address in the Stack Pointer, SP, then loads the high order byte of PC with the data at the memory address immediately following the one held in SP. The data in SP is then incremented twice.

RET *f*

Opcode	Instruction	Operation
——	**RET *f***	**If {*f*} $PC_{(low)}$ = (SP); $PC_{(high)}$ = (SP + 1); SP = SP + 2**
C0	RET NZ	If {NZ} $PC_{(low)}$ = (SP); $PC_{(high)}$ = (SP + 1); SP = SP + 2
C8	RET Z	If {Z} $PC_{(low)}$ = (SP); $PC_{(high)}$ = (SP + 1); SP = SP + 2
D0	RET NC	If {NC} $PC_{(low)}$ = (SP); $PC_{(high)}$ = (SP + 1); SP = SP + 2
D8	RET C	If {C} $PC_{(low)}$ = (SP); $PC_{(high)}$ = (SP + 1); SP = SP + 2
E0	RET LZ	If {LZ} $PC_{(low)}$ = (SP); $PC_{(high)}$ = (SP + 1); SP = SP + 2
E8	RET LO	If {LO} $PC_{(low)}$ = (SP); $PC_{(high)}$ = (SP + 1); SP = SP + 2
F0	RET P	If {P} $PC_{(low)}$ = (SP); $PC_{(high)}$ = (SP + 1); SP = SP + 2
F8	RET M	If {M} $PC_{(low)}$ = (SP); $PC_{(high)}$ = (SP + 1); SP = SP + 2
Clocking: 2; 8 (2,1,2,2,1)		

Flags							
S	**Z**				**L/V**		**C**
-	-				-		-

ALTD		
F	**R**	**SP**

I/O	
S	**D**

Description

If the condition *f* is false, then the instruction is ignored. If the condition *f* is true, then the instruction loads the low order byte of the Program Counter, PC, with the data at the memory address in the Stack Pointer, SP, then loads the high order byte of PC with the data at the memory address immediately following the one held in SP and the data in SP is then incremented twice.

The condition *f* is one of the following:

- **NZ** zero flag not set
- **Z** zero flag set
- **NC** carry flag not set
- **C** carry flag set
- **LZ/NV** Logic Zero/Overflow flag is not set
- **LO/V** Logic Zero/Overflow flag is set
- **P** sign flag not set

RETI NP

Opcode	Instruction	Clocks	Operation
ED 4D	RETI	12 (2,2,1,2,2,2,1)	IP = (SP); $PC_{(low)}$ = (SP+1); $PC_{(high)}$ = (SP + 2); SP = SP + 3

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

Loads the Interrupt Priority register, IP, with the data whose address is in the Stack Pointer, SP. Then loads the low order byte of the Program Counter, PC, with the data whose address is 1 higher than the data in SP and loads the high order byte of the PC with the data whose address is two higher than the data in the SP. The data in the SP is then incremented three times.

RL (HL)
RL (IX+*d*)
RL (IY+*d*)

Opcode	Instruction	Clocks	Operation
CB 16	RL (HL)	10 (2,2,1,2,3)	{CF,(HL)} = {(HL),CF}
DD CB *d* 16	RL (IX+*d*)	13 (2,2,2,2,2,3)	{CF,(IX + *d*)} = {(IX + *d*),CF}
FD CB *d* 16	RL (IY+*d*)	13 (2,2,2,2,2,3)	{CF,(IY + *d*)} = {(IY + *d*),CF}

Flags							
S	Z				L/V		C
•	•				L		•

ALTD		
F	R	SP
•		

I/O	
S	D
•	•

Description

Rotates to the left with the Carry Flag, CF, the data whose address is:

- the data in word register HL, or
- the sum of the data in index register IX and a displacement *d*, or
- the sum of the data in index register IY and a displacement *d*.

Bits 0 through 6 move to the next highest-order bit position (bit 0 moves to bit 1, etc.) while the CF moves to bit 0 and bit 7 moves to the CF. See Figure 1 below.

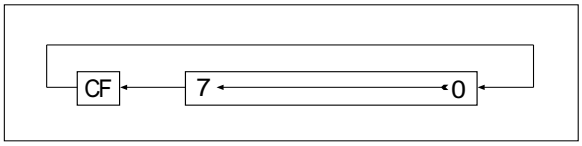


Figure 1: The bit logic of the RL instruction.

Example

If the HL contains 0x4545, the byte in the memory location 0x4545 is 0110 1010, and the CF is set, then after the execution of the operation

```
RL (HL)
```

The byte in memory location 0x4545 will contain 1101 0101 and the CF will be reset.

RL DE

N

Opcode	Instruction	Clocks	Operation
F3	RL DE	2	{CF,DE} = {DE,CF}

Flags							
S	Z				L/V		C
•	•				L		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Rotates to the left with the Carry Flag, CF, the contents of register DE. Each bit in the register moves to the next highest-order bit position (bit 0 moves to bit 1, etc.) while the CF moves to bit 0 and bit 15 moves to the CF. See Figure 1 on page 80.

RL *r*

Opcode	Instruction	Clocks	Operation
——	**RL *r***	**4 (2,2)**	**{CF,*r*} = {*r*,CF}**
CB 17	RL A	4 (2,2)	{CF,A} = {A,CF}
CB 10	RL B	4 (2,2)	{CF,B} = {B,CF}
CB 11	RL C	4 (2,2)	{CF,C} = {C,CF}
CB 12	RL D	4 (2,2)	{CF,D} = {D,CF}
CB 13	RL E	4 (2,2)	{CF,E} = {E,CF}
CB 14	RL H	4 (2,2)	{CF,H} = {H,CF}
CB 15	RL L	4 (2,2)	{CF,L} = {L,CF}

Flags							
S	Z				L/V		C
•	•				L		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Rotates to the left with the Carry Flag, CF, the contents of the register *r* (any of the register A, B, C, D, E, H, or L). Each bit in the register moves to the next highest-order bit position (bit 0 moves to bit 1, etc.) while the CF moves to bit 0 and bit 7 moves to the CF. See Figure 1 on page 80.

RLA

Opcode	Instruction	Clocks	Operation
17	RLA	2	{CF,A} = {A,CF}

Flags							
S	Z				L/V		C
-	-				-		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Rotates to the left with the Carry Flag, CF, the contents of the Accumulator. Each bit in the register moves to the next highest-order bit position (bit 0 moves to bit 1, etc.) while the CF moves to bit 0 and bit 7 moves to the CF. See Figure 1 on page 80.

RLC (HL)
RLC (IX+*d*)
RLC (IY+*d*)

Opcode	Instruction	Clk	Operation
CB 06	RLC (HL)	10*	(HL) = {(HL)[6,0],(HL)[7]}; CF = (HL)[7]
DD CB *d* 06	RLC (IX+*d*)	13**	(IX + *d*) = {(IX + *d*)[6,0],(IX + *d*)[7]}; CF = (IX+*d*)[7]
FD CB *d* 06	RLC (IY+*d*)	13**	(IY + *d*) = {(IY + *d*)[6,0],(IY + *d*)[7]}; CF = (IY + *d*)[7]
Clk: Clocking: *10 (2,2,1,2,3) **13 (2,2,2,2,2,3)			

Flags							
S	Z				L/V		C
•	•				L		•

ALTD		
F	R	SP
•		

I/O	
S	D
•	•

Description

Rotates to the left the data whose address is:

- the data in word register HL, or
- the sum of the data in index register IX and a displacement *d*, or
- the sum of the data in index register IY and a displacement *d*.

Each bit in the register moves to the next highest-order bit position (bit 0 moves to bit 1, etc.) while bit 7 moves to both bit 0 and the CF. See Figure 2 below.

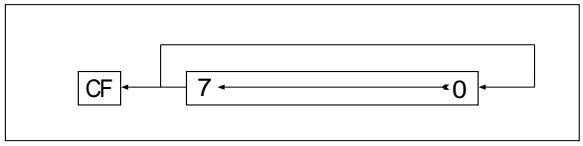


Figure 2: The bit logic of the RLC instruction.

Example

If the HL contains 0x4545, the byte in the memory location 0x4545 is 0110 1010, and the CF is set, then after the execution of the operation:

```
RLC (HL)
```

the byte in memory location 0x4545 will contain 1101 0100 and the CF will be reset.

RLC *r*

Opcode	Instruction	Clocks	Operation
——	**RLC *r***	**4 (2,2)**	***r* = {*r*[6,0],*r*[7]}; CF = *r*[7]**
CB 07	RLC A	4 (2,2)	A = {A[6,0],A[7]}; CF = A[7]
CB 00	RLC B	4 (2,2)	B = {B[6,0],B[7]}; CF = B[7]
CB 01	RLC C	4 (2,2)	C = {C[6,0],C[7]}; CF = C[7]
CB 02	RLC D	4 (2,2)	D = {D[6,0],D[7]}; CF = D[7]
CB 03	RLC E	4 (2,2)	E = {E[6,0],E[7]}; CF = E[7]
CB 04	RLC H	4 (2,2)	H = {H[6,0],H[7]}; CF = H[7]
CB 05	RLC L	4 (2,2)	L = {L[6,0],L[7]}; CF = L[7]

Flags							
S	Z				L/V		C
•	•				L		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Rotates to the left the data in the register *r* (any of the register A, B, C, D, E, H, or L). Each bit in the register moves to the next highest-order bit position (bit 0 moves to bit 1, etc.) while bit 7 moves to both bit 0 and the CF. See Figure 2 on page 83.

RLCA

Opcode	Instruction	Clocks	Operation
07	RLCA	2	A = {A[6,0],A[7]}; CF = A[7]

Flags							
S	Z				L/V		C
-	-				-		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Rotates to the left the data in the Accumulator. Each bit in the register moves to the next highest-order bit position (bit 0 moves to bit 1, etc.) while bit 7 moves to both bit 0 and the CF. See Figure 2 on page 83.

RR (HL)
RR (IX+*d*)
RR (IY+*d*)

Opcode	Instruction	Clocks	Operation
CB 1E	RR (HL)	10 (2,2,1,2,3)	{(HL),CF} = {CF,(HL)}
DD CB *d* 1E	RR (IX+*d*)	13 (2,2,2,2,2,3)	{(IX+*d*),CF} = {CF,(IX+*d*)}
FD CB *d* 1E	RR (IY+*d*)	13 (2,2,2,2,2,3)	{(IY+*d*),CF} = {CF,(IY+*d*)}

Flags							
S	Z				L/V		C
•	•				L		•

ALTD		
F	R	SP
•		

I/O	
S	D
•	•

Description

Rotates to the right with the Carry Flag, CF, the data whose address is:

- the data in word register HL, or
- the sum of the data in index register IX and a displacement *d*, or
- the sum of the data in index register IY and a displacement *d*.

Bit 0 moves to the CF, bits 1 through 7 move to the next lowest-order bit position, and the CF moves to bit 7. See Figure 3 below.

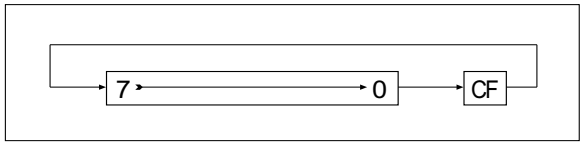


Figure 3: The bit logic for the RR instruction.

RR DE N
RR HL N

Opcode	Instruction	Clocks	Operation
FB	RR DE	2	{DE,CF} = {CF,DE}
FC	RR HL	2	{HL,CF} = {CF,HL}

Flags							
S	Z				L/V		C
•	•				L		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Rotates to the right with the Carry Flag, CF, the data in word register DE or HL. Bit 0 moves to the CF, bits 1 through 15 move to the next lowest-order bit position, and the CF moves to bit 15. See Figure 3 on page 85.

RR IX N
RR IY N

Opcode	Instruction	Clocks	Operation
DD FC	RR IX	4 (2,2)	{IX,CF} = {CF,IX}
FD FC	RR IY	4 (2,2)	{IY,CF} = {CF,IY}

Flags							
S	Z				L/V		C
•	•				L		•

ALTD		
F	R	SP
•		

I/O	
S	D

Description

Rotates to the right with the Carry Flag, CF, the data in index register IX or IY. Bit 0 moves to the CF, bits 1 through 15 move to the next lowest-order bit position, and the CF moves to bit 15. See Figure 3 on page 85.

RR *r*

Opcode	Instruction	Clocks	Operation
——	**RR *r***	**4 (2,2)**	**{*r*,CF} = {CF,*r*}**
CB 1F	RR A	4 (2,2)	{A,CF} = {CF,A}
CB 18	RR B	4 (2,2)	{B,CF} = {CF,B}
CB 19	RR C	4 (2,2)	{C,CF} = {CF,C}
CB 1A	RR D	4 (2,2)	{D,CF} = {CF,D}
CB 1B	RR E	4 (2,2)	{E,CF} = {CF,E}
CB 1C	RR H	4 (2,2)	{H,CF} = {CF,H}
CB 1D	RR L	4 (2,2)	{L,CF} = {CF,L}

Flags							
S	Z				L/V		C
•	•				L		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Rotates to the right with the Carry Flag, CF, the data in register *r* (any of the registers A, B, C, D, E, H, or L). Bit 0 moves to the CF, bits 1 through 7 move to the next lowest-order bit position, and the CF moves to bit 7. See Figure 3 on page 85.

RRA

Opcode	Instruction	Clocks	Operation
1F	RRA	2	{A,CF} = {CF,A}

Flags							
S	Z				L/V		C
-	-				-		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Rotates to the right with the Carry Flag, CF, the data in the Accumulator. Bit 0 moves to the CF, bits 1 through 15 move to the next lowest-order bit position, and the CF moves to bit 15. See Figure 3 on page 85.

RRC (HL)
RRC (IX+*d*)
RRC (IY+*d*)

Opcode	Instruction	Clocks	Operation
CB 0E	RRC (HL)	10 (2,2,1,2,3)	(HL) = {(HL)[0],(HL)[7,1]}; CF = (HL)[0]
DD CB *d* 0E	RRC (IX+*d*)	13 (2,2,2,2,2,3)	(IX + *d*) = {(IX + *d*)[0], (IX + *d*)[7,1]}; CF = (IX + *d*)[0]
FD CB *d* 0E	RRC (IY+*d*)	13 (2,2,2,2,2,3)	(IY + *d*) = {(IY + *d*)[0], (IY + *d*)[7,1]}; CF = (IY + *d*)[0]

Flags							
S	Z				L/V		C
•	•				L		•

ALTD		
F	R	SP
•		

I/O	
S	D
•	•

Description

Rotates to the right the data whose address is:

- the data in word register HL, or
- the sum of the data in index register IX and a displacement *d*, or
- the sum of the data in index register IY and a displacement *d*.

Each bit in the register moves to the next lowest-order bit position (bit 7 moves to bit 6, etc.) while bit 0 moves to both bit 7 and the CF. See Figure 4 below.

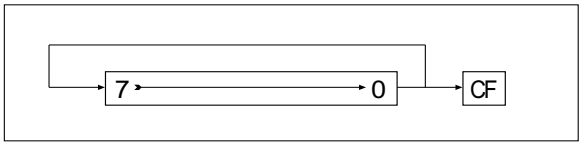


Figure 4: The bit logic of the RRC instruction.

RRC *r*

Opcode	Instruction	Clocks	Operation
——	**RRC *r***	**4 (2,2)**	***r* = {*r*[0],*r*[7,1]}; CF = *r*[0]**
CB 0F	RRC A	4 (2,2)	A = {A[0],A[7,1]}; CF = A[0]
CB 08	RRC B	4 (2,2)	B = {B[0],B[7,1]}; CF = B[0]
CB 09	RRC C	4 (2,2)	C = {C[0],C[7,1]}; CF = C[0]
CB 0A	RRC D	4 (2,2)	D = {D[0],D[7,1]}; CF = D[0]
CB 0B	RRC E	4 (2,2)	E = {E[0],E[7,1]}; CF = E[0]
CB 0C	RRC H	4 (2,2)	H = {H[0],H[7,1]}; CF = H[0]
CB 0D	RRC L	4 (2,2)	L = {L[0],L[7,1]}; CF = L[0]

Flags							
S	Z				L/V		C
•	•				L		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Rotates to the right the data in the register *r* (any of the registers A, B, C, D, E, H, or L). Each bit in the register moves to the next lowest-order bit position (bit 7 moves to bit 6, etc.) while bit 0 moves to both bit 7 and the CF. See Figure 4 on page 88.

RRCA

Opcode	Instruction	Clocks	Operation
0F	RRCA	2	A = {A[0],A[7,1]}; CF = A[0]

Flags							
S	Z				L/V		C
-	-				-		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Rotates to the right the data in the Accumulator. Each bit in the register moves to the next lowest-order bit position (bit 7 moves to bit 6, etc.) while bit 0 moves to both bit 7 and the CF. See Figure 4 on page 88.

Description

Pushes the current Program Counter, PC, onto the stack and then resets the PC to the interrupt vector address represented by **IIR:v**, where **IIR** is the address of the interrupt table and **v** is the offset into the table. The address of the vector table can be read and set by the instructions LD A,IIR and LD IIR,A respectively, where A is the upper nibble of the 16 bit vector table address. The vector table is always on a 100h boundary.

The push is accomplished by first loading the high-order byte of the PC into the memory location with the address 1 less than the number in the Stack Pointer, SP. Then the low-order byte of the PC is loaded into the memory location with the address two less than the number in the SP. The value in the SP is then decremented twice.

The PC is reset by loading it with the address it is

SBC A,(HL)
SBC (IX+*d*)
SBC (IY+*d*)

Opcode	Instruction	Clocks	Operation
9E	SBC A,(HL)	5 (2,1,2)	A = A - (HL) - CF
DD 9E *d*	SBC (IX+*d*)	9 (2,2,2,1,2)	A = A - (IX + *d*) - CF
FD 9E *d*	SBC (IY+*d*)	9 (2,2,2,1,2)	A = A - (IY + *d*) - CF

Flags							
S	Z				L/V		C
•	•				V		•

ALTD		
F	R	SP
•	•	

I/O	
S	D
•	

Description

Subtracts the Carry Flag, CF, and the data whose address is:

- the data in word register HL, or
- the sum of the data in index register IX and a displacement *d*, or
- the sum of the data in index register IY and a displacement *d*

from the data in the Accumulator. The result is stored in the Accumulator.

These operations output an inverted carry:

- The Carry Flag is set if the Accumulator is less than the data being subtracted from it.
- The Carry Flag is cleared if the Accumulator is greater than the data being subtracted from it.

SBC A,*n*
SBC A,*r*

Opcode	Instruction	Clocks	Operation
DE *n*	SBC A,*n*	4 (2,2)	A = A - *n* - CF
——	**SBC A,*r***	**2**	**A = A - *r* - CF**
9F	SBC A,A	2	A = A - A - CF
98	SBC A,B	2	A = A - B - CF
99	SBC A,C	2	A = A - C - CF
9A	SBC A,D	2	A = A - D - CF
9B	SBC A,E	2	A = A - E - CF
9C	SBC A,H	2	A = A - H - CF
9D	SBC A,L	2	A = A - L - CF

Flags							
S	Z				L/V		C
•	•				V		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

- **SBC A,*n*:** Subtracts the Carry Flag, CF, and the 8-bit constant *n* from the data in the Accumulator.
- **SBC A,*r*:** Subtracts the Carry Flag, CF, and the data in the register *r* (any of the registers A, B, C, D, E, H, or L) from the data in the Accumulator.

The difference is stored in the Accumulator.

These operations output an inverted carry:

- The Carry Flag is set if the Accumulator is less than the data being subtracted from it.
- The Carry Flag is cleared if the Accumulator is greater than the data being subtracted from it.

SBC HL,*ss*

Opcode	Instruction	Clocks	Operation
——	**SBC HL,*ss***	**4 (2,2)**	**HL = HL - *ss* - CF**
ED 42	SBC HL,BC	4 (2,2)	HL = HL - BC - CF
ED 52	SBC HL,DE	4 (2,2)	HL = HL - DE - CF
ED 62	SBC HL,HL	4 (2,2)	HL = HL - HL - CF
ED 72	SBC HL,SP	4 (2,2)	HL = HL - SP - CF

Flags							
S	**Z**				**L/V**		**C**
•	•				V		•

ALTD		
F	**R**	**SP**
•	•	

I/O	
S	**D**

Description

Subtracts the Carry Flag, CF, and the data in word register *ss* (any of the word registers BC, DE, HL, or SP) from the data in word register HL. The difference is stored in HL.

These operations output an inverted carry:

- The Carry Flag is set if the Accumulator is less than the data being subtracted from it.
- The Carry Flag is cleared if the Accumulator is greater than the data being subtracted from it.

SCF

Opcode	Instruction	Clocks	Operation
37	SCF	2	CF = 1

Flags							
S	**Z**				**L/V**		**C**
-	-				-		1

ALTD		
F	**R**	**SP**
•		

I/O	
S	**D**

Description

Sets the Carry Flag, CF.

SET *b*,(HL)
SET *b*,(IX+*d*)
SET *b*,(IY+*d*)

Opcode	Instruction	Clocks	Operation
	SET *b*,(HL)	10*	(HL) = (HL) | bit
CB C6	SET bit 0,(HL)	10*	(HL) = (HL) | bit 0
CB CE	SET bit 1,(HL)	10*	(HL) = (HL) | bit 1
CB D6	SET bit 2,(HL)	10*	(HL) = (HL) | bit 2
CB DE	SET bit 3,(HL)	10*	(HL) = (HL) | bit 3
CB E6	SET bit 4,(HL)	10*	(HL) = (HL) | bit 4
CB EE	SET bit 5,(HL)	10*	(HL) = (HL) | bit 5
CB F6	SET bit 6,(HL)	10*	(HL) = (HL) | bit 6
CB FE	SET bit 7,(HL)	10*	(HL) = (HL) | bit 7
	SET *b*,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) | bit
DD CB *d* C6	SET bit 0,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) | bit 0
DD CB *d* CE	SET bit 1,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) | bit 1
DD CB *d* D6	SET bit 2,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) | bit 2
DD CB *d* DE	SET bit 3,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) | bit 3
DD CB *d* E6	SET bit 4,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) | bit 4
DD CB *d* EE	SET bit 5,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) | bit 5
DD CB *d* F6	SET bit 6,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) | bit 6
DD CB *d* FE	SET bit 7,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) | bit 7
	SET *b*,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) | bit
DD CB *d* C6	SET bit 0,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) | bit 0
DD CB *d* CE	SET bit 1,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) | bit 1
DD CB *d* D6	SET bit 2,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) | bit 2
DD CB *d* DE	SET bit 3,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) | bit 3
DD CB *d* E6	SET bit 4,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) | bit 4
DD CB *d* EE	SET bit 5,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) | bit 5
DD CB *d* F6	SET bit 6,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) | bit 6
DD CB *d* FE	SET bit 7,(IX+*d*)	13**	(IX + *d*) = (IX + *d*) | bit 7
Clocking: *10 (2,2,1,2,3) **13 (2,2,2,2,2,3)			

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D
•	•

Description

Sets bit *b* (any of the bits 0, 1, 2, 3, 4, 5, 6, or 7) of the byte whose address is

- the data in word register HL, or
- the sum of the data in index register IX and a displacement *d*, or
- the sum of the data in index register IY and a displacement *d*.

SET *b,r*

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

Flags							
S	**Z**				**L/V**		**C**
-	-				-		-

ALTD		
F	**R**	**SP**
	•	

I/O	
S	**D**

Description

Sets bit *b* (any of the bits 0, 1, 2, 3, 4, 5, 6, or 7) of the data in register *r* (any of the registers A, B, C, D, E, H, or L).

SLA (HL)
SLA (IX+*d*)
SLA (IY+*d*)

Opcode	Instruction	Clocks	Operation
CB 26	SLA (HL)	10*	(HL) = {(HL)[6,0],0}; CF = (HL)[7]
DD CB *d* 26	SLA (IX+*d*)	13**	(IX + *d*) = {(IX + *d*)[6,0],0}; CF = (IX + *d*)[7]
FD CB *d* 26	SLA (IY+*d*)	13**	(IY + *d*) = {(IY + *d*)[6,0],0}; CF = (IY + *d*)[7]
Clocking: *10 (2,2,1,2,3) **13 (2,2,2,2,2,3)			

Flags							
S	Z				L/V		C
•	•				L		•

ALTD		
F	R	SP
•		

I/O	
S	D
•	•

Description

Arithmetically shifts to the left the bits of the data whose address is

- the data in word register HL, or
- the sum of the data in index register IX and a displacement *d*, or
- the sum of the data in index register IY and a displacement *d*.

Bits 0 through 6 are each shifted to the next highest-order bit position (bit 0 moves to bit 1, etc.). Bit 7 is shifted to the Carry Flag, CF. Bit 0 is reset. See Figure 5 below.

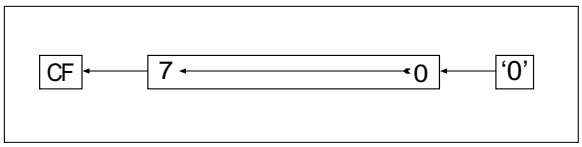


Figure 5: The bit logic of the SLA instruction.

SLA *r*

Opcode	Instruction	Clocks	Operation
——	**SLA *r***	**4 (2,2)**	***r* = {*r*[6,0],0}; CF = *r*[7]**
CB 27	SLA A	4 (2,2)	A = {A[6,0],0}; CF = A[7]
CB 20	SLA B	4 (2,2)	B = {B[6,0],0}; CF = B[7]
CB 21	SLA C	4 (2,2)	C = {C[6,0],0}; CF = C[7]
CB 22	SLA D	4 (2,2)	D = {D[6,0],0}; CF = D[7]
CB 23	SLA E	4 (2,2)	E = {E[6,0],0}; CF = E[7]
CB 24	SLA H	4 (2,2)	H = {H[6,0],0}; CF = H[7]
CB 25	SLA L	4 (2,2)	L = {L[6,0],0}; CF = L[7]

Flags							
S	**Z**				**L/V**		**C**
•	•				L		•

ALTD		
F	**R**	**SP**
•	•	

I/O	
S	**D**

Description

Arithmetically shifts to the left the bits of the data in register *r* (any of A, B, C, D, E, H, or L). Bits 0 through 6 are each shifted to the next highest-order bit position (bit 0 moves to bit 1, etc.). Bit 7 is shifted to the Carry Flag, CF. Bit 0 is reset. See Figure 5 on page 96.

SRA (HL)
SRA (IX+*d*)
SRA (IY+*d*)

Opcode	Instruction	Clocks	Operation
CB 2E	SRA (HL)	10*	(HL) = {(HL)[7],(HL)[7,1]}; CF = (HL)[0]
DD CB *d* 2E	SRA (IX+*d*)	13**	(IX + *d*) = {(IX + *d*)[7],(IX + *d*)[7,1]}; CF = (IX + *d*)[0]
FD CB *d* 2E	SRA (IY+*d*)	13**	(IY +*d*) = {(IY + *d*)[7],(IY + *d*)[7,1]}; CF = (IY + *d*)[0]
Clocking: *10 (2,2,1,2,3) **13 (2,2,2,2,2,3)			

Flags							
S	Z				L/V		C
•	•				L		•

ALTD		
F	R	SP
•		

I/O	
S	D
•	•

Description

Arithmetically shifts to the right the bits in the data whose address is

- the data in word register HL, or
- the sum of the data in index register IX and a displacement *d*, or
- the sum of the data in index register IY and a displacement *d*.

Bits 7 through 1 are shifted to the next lowest-order bit position (bit 7 is shifted to bit 6, etc.). Bit 7 is also copied to itself. Bit 0 is shifted to the Carry Flag, CF. See Figure 6 below.

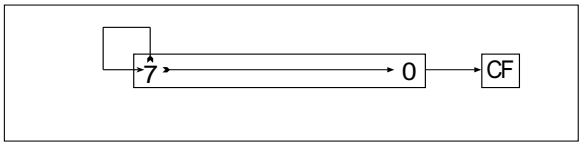


Figure 6: The bit logic of the SRA instruction.

SRA *r*

Opcode	Instruction	Clocks	Operation
——	**SRA *r***	**4 (2,2)**	***r* = {*r*[7],*r*[7,1]}; CF = *r*[0]**
CB 2F	SRA A	4 (2,2)	A = {A[7],A[7,1]}; CF = A[0]
CB 28	SRA B	4 (2,2)	B = {B[7],B[7,1]}; CF = B[0]
CB 29	SRA C	4 (2,2)	C = {C[7],C[7,1]}; CF = C[0]
CB 2A	SRA D	4 (2,2)	D = {D[7],D[7,1]}; CF = D[0]
CB 2B	SRA E	4 (2,2)	E = {E[7],E[7,1]}; CF = E[0]
CB 2C	SRA H	4 (2,2)	H = {H[7],H[7,1]}; CF = H[0]
CB 2D	SRA L	4 (2,2)	L = {L[7],L[7,1]}; CF = L[0]

Flags							
S	**Z**				**L/V**		**C**
•	•				L		•

ALTD		
F	**R**	**SP**
•	•	

I/O	
S	**D**

Description

Arithmetically shifts to the right the bits in the register *r* (any of the registers A, B, C, D, E, H, or L). Bits 7 through 1 are shifted to the next lowest-order bit position (bit 7 is shifted to bit 6, etc.). Bit 7 is also copied to itself. Bit 0 is shifted to the Carry Flag, CF. See Figure 6 on page 98.

SRL (HL)
SRL (IX+*d*)
SRL (IY+*d*)

Opcode	Instruction	Clocks	Operation
CB 3E	SRL (HL)	10*	(HL) = {0,(HL)[7,1]}; CF = (HL)[0]
DD CB *d* 3E	SRL (IX+*d*)	13**	(IX + *d*) = {0,(IX + *d*)[7,1]}; CF = (IX + *d*)[0]
FD CB *d* 3E	SRL (IY+*d*)	13**	(IY + *d*) = {0,(IY + *d*)[7,1]}; CF = (IY + *d*)[0]
Clocking: *10 (2,2,1,2,3) **13 (2,2,2,2,2,3)			

Flags							
S	Z				L/V		C
•	•				L		•

ALTD		
F	R	SP
•		

I/O	
S	D
•	•

Description

Logically shifts to the right the bits of the data whose address is

- the data in word register HL, or
- the sum of the data in index register IX and a displacement *d*, or
- the sum of the data in index register IY and a displacement *d*.

Each bit is shifted to the next lowest-order bit position (Bit 7 shifts to bit 6, etc.) Bit 0 shift to the Carry Flag, CF. Bit 7 is reset. See Figure 7 below.

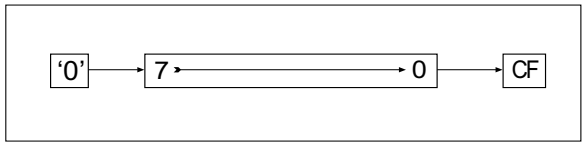


Figure 7: The bit logic of the SRL instruction.

SRL *r*

Opcode	Instruction	Clocks	Operation
——	**SRL *r***	**4 (2,2)**	***r* = {0,*r*[7,1]}; CF = *r*[0]**
CB 3E	SRL A	4 (2,2)	A = {0,A[7,1]}; CF = A[0]
CB 38	SRL B	4 (2,2)	B = {0,B[7,1]}; CF = B[0]
CB 39	SRL C	4 (2,2)	C = {0,C[7,1]}; CF = C[0]
CB 3A	SRL D	4 (2,2)	D = {0,D[7,1]}; CF = D[0]
CB 3B	SRL E	4 (2,2)	E = {0,E[7,1]}; CF = E[0]
CB 3C	SRL H	4 (2,2)	H = {0,H[7,1]}; CF = H[0]
CB 3D	SRL L	4 (2,2)	L = {0,L[7,1]}; CF = L[0]

Flags							
S	**Z**				**L/V**		**C**
•	•				L		•

ALTD		
F	**R**	**SP**
•	•	

I/O	
S	**D**

Description

Logically shifts to the right the bits in the register *r* (any of the registers A, B, C, D, E, H, or L). Each bit is shifted to the next lowest-order bit position (Bit 7 shifts to bit 6, etc.) Bit 0 shift to the Carry Flag, CF. Bit 7 is reset. See Figure 7 on page 100.

SUB (HL)
SUB (IX+*d*)
SUB (IY+*d*)

Opcode	Instruction	Clocks	Operation
96	SUB (HL)	5 (2,1,2)	A = A - (HL)
DD 96 *d*	SUB (IX+*d*)	9 (2,2,2,1,2)	A = A - (IX + *d*)
FD 96 *d*	SUB (IY+*d*)	9 (2,2,2,1,2)	A = A - (IY + *d*)

Flags							
S	Z				L/V		C
•	•				V		•

ALTD		
F	R	SP
•	•	

I/O	
S	D
•	

Description

Subtracts from the data in the Accumulator the data whose address is

- the data in word register HL,or
- the sum of the data in index register IX and a displacement *d*, or
- the sum of the data in index register IY and a displacement *d*.

The result is stored in the Accumulator.

SUB *n*

Opcode	Instruction	Clocks	Operation
D6 *n*	SUB *n*	4 (2,2)	A = A - *n*

Flags							
S	Z				L/V		C
•	•				V		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Subtracts from the data in the Accumulator the 8-bit constant *n*. The result is stored in the Accumulator.

SUB *r*

Opcode	Instruction	Clocks	Operation
——	**SUB *r***	**2**	**A = A - *r***
97	SUB A	2	A = A - A
90	SUB B	2	A = A - B
91	SUB C	2	A = A - C
92	SUB D	2	A = A - D
93	SUB E	2	A = A - E
94	SUB H	2	A = A - H
95	SUB L	2	A = A - L

Flags							
S	Z				L/V		C
•	•				V		•

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Subtracts from the data in the Accumulator the data in the register *r* (any of the registers A, B, C, D, E, H, or L). The result is stored in the Accumulator.

XOR (HL)
XOR (IX+*d*)
XOR (IY+*d*)

Opcode	Instruction	Clocks	Operation
AE	XOR (HL)	5 (2,1,2)	A = [A & ~(HL)] | [~A & (HL)]
DD AE *d*	XOR (IX+*d*)	9 (2,2,2,1,2)	A = [A & ~(IX + *d*)] | [~A & (IX + *d*)]
FD AE *d*	XOR (IY+*d*)	9 (2,2,2,1,2)	A = [A & ~(IY + *d*)] | [~A & (IY + *d*)]

Flags							
S	Z				L/V		C
•	•				L		0

ALTD		
F	R	SP
•	•	

I/O	
S	D
•	

Description

Performs an exclusive OR operation between the data in the Accumulator and the data whose address is:

- the data in word register HL, or
- the sum of the data in index register IX and a displacement *d*, or
- the sum of the data in index register IY and a displacement *d*.

The corresponding bits of each byte are compared (i.e., the bit 1 of both bytes are compared, the bit 2 of both bytes are compared, etc.) and the associated bit in the result byte is set if and only if one of the two compared bits is set. The result is stored in the Accumulator.

Example

If the HL contains 0x4000 and the memory location 0x4000 contains the byte 1001 0101 and the Accumulator contains the byte 0101 0011 then the execution of the instruction

```
XOR (HL)
```

would result in the byte in the Accumulator becoming 1100 0110.

XOR *n*

Opcode	Instruction	Clocks	Operation
EE *n*	XOR *n*	4 (2,2)	A = [A & ~*n*] | [~A & *n*]

Flags							
S	Z				L/V		C
•	•				L		0

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Performs an exclusive OR operation between the byte in the Accumulator and the 8-bit constant *n*. The corresponding bits of each byte are compared (i.e., the bit 1 of both bytes are compared, the bit 2 of both bytes are compared, etc.) and the associated bit in the result byte is set if and only if one of the two compared bits is set. The result is stored in the Accumulator.

XOR *r*

Opcode	Instruction	Clocks	Operation
——	**XOR *r***	**2**	**A = [A & ~*r*] | [~A & *r*]**
AF	XOR A	2	A = [A & ~A] | [~A & A]
A8	XOR B	2	A = [A & ~B] | [~A & B]
A9	XOR C	2	A = [A & ~C] | [~A & C]
AA	XOR D	2	A = [A & ~D] | [~A & D]
AB	XOR E	2	A = [A & ~E] | [~A & E]
AC	XOR H	2	A = [A & ~H] | [~A & H]
AD	XOR L	2	A = [A & ~L] | [~A & L]

Flags							
S	Z				L/V		C
•	•				L		0

ALTD		
F	R	SP
•	•	

I/O	
S	D

Description

Performs an exclusive OR operation between the byte in the Accumulator and the register *r* (any of the registers A, B, C, D, E, H, or L). The corresponding bits of each byte are compared (i.e., the bit 1 of both bytes are compared, the bit 2 of both bytes are compared, etc.) and the associated bit in the result byte is set if and only if one of the two compared bits is set. The result is stored in the Accumulator.

5. Quick Reference Table

Key

- **`Instruction`**: The mnemonic syntax of the instruction.
- **`Opcode`**: The binary bytes that represent the instruciton.
- **`Clock cycles`**: The number of clock cycles that the instrucion takes to complete. The numbers in parenthesis are a breakdown of the total clocks. See Table 1 on page 5 for more details.
- **`A`**: How the ALTD prefix affects the intruction. See Table 2 on page 6.
- **`I`**: How the IOI or IOE prefixes affect the instruction. See Table 3 on page 6. A “b” in this column indicates that the prefix applies to both source and destination.
- **`S`**; **`Z`**; **`LV`**; **`C`**: These columns denote how the instruction affects the flags. See Table 4 on page 6.
- **`Operation`**: A symbolic representation of the operation performed.
- **`N/M/P`**: An “N” in this column indicates that the instruction has been added to the Z180 instruction set by the Rabbit 2000. An “M” indicates that this instruction is from the Z180, but has been modified. A “P” indicates a privileged instruction.

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Clock cycles	A	I	S	Z	LV	C	Operation	N/M/P
ADC A,(HL)	10001110				5 (2,1,2)	fr	s	*	*	V	*	A = A + (HL) + CF	
ADC A,(IX+d)	11011101	10001110	----d---		9 (2,2,2,1,2)	fr	s	*	*	V	*	A = A + (IX+d) + CF	
ADC A,(IY+d)	11111101	10001110	----d---		9 (2,2,2,1,2)	fr	s	*	*	V	*	A = A + (IY+d) + CF	
ADC A,n	11001110	----n---			4 (2,2)	fr		*	*	V	*	A = A + n + CF	
ADC A,r	10001-r-				2	fr		*	*	V	*	A = A + r + CF	
ADC HL,ss	11101101	01ss1010			4 (2,2)	fr		*	*	V	*	HL = HL + ss + CF	
ADD A,(HL)	10000110				5 (2,1,2)	fr	s	*	*	V	*	A = A + (HL)	
ADD A,(IX+d)	11011101	10000110	----d---		9 (2,2,2,1,2)	fr	s	*	*	V	*	A = A + (IX+d)	
ADD A,(IY+d)	11111101	10000110	----d---		9 (2,2,2,1,2)	fr	s	*	*	V	*	A = A + (IY+d)	
ADD A,n	11000110	----n---			4 (2,2)	fr		*	*	V	*	A = A + n	
ADD A,r	10000-r-				2	fr		*	*	V	*	A = A + r	
ADD HL,ss	00ss1001				2	fr		-	-	-	*	HL = HL + ss	
ADD IX,xx	11011101	00xx1001			4 (2,2)	f		-	-	-	*	IX = IX + xx	
ADD IY,yy	11111101	00yy1001			4 (2,2)	f		-	-	-	*	IY = IY + yy	
ADD SP,d	00100111	----d---			4 (2,2)	f		-	-	-	*	SP = SP + d	N
ALTD	01110110				2			-	-	-	-	alternate register destination for next instruction	N
AND (HL)	10100110				5 (2,1,2)	fr	s	*	*	L	0	A = A & (HL)	
AND (IX+d)	11011101	10100110	----d---		9 (2,2,2,1,2)	fr	s	*	*	L	0	A = A & (IX+d)	
AND (IY+d)	11111101	10100110	----d---		9 (2,2,2,1,2)	fr	s	*	*	L	0	A = A & (IY+d)	
AND HL,DE	11011100				2	fr		*	*	L	0	HL = HL & DE	N
AND IX,DE	11011101	11011100			4 (2,2)	f		*	*	L	0	IX = IX & DE	N
AND IY,DE	11111101	11011100			4 (2,2)	f		*	*	L	0	IY = IY & DE	N
AND n	11100110	----n---			4 (2,2)	fr		*	*	L	0	A = A & n	
AND r	10100-r-				2	fr		*	*	L	0	A = A & r	
BIT b,(HL)	11001011	01-b-110			7 (2,2,1,2)	f	s	-	*	-	-	(HL) & bit	P

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Clock cycles	A	I	S	Z	LV	C	Operation	N/M/P
BIT b,(IX+d)	11011101	11001011	----d---	01-b-110	10 (2,2,2,2,2)	f	s	-	*	-	-	(IX+d) & bit	
BIT b,(IY+d)	11111101	11001011	----d---	01-b-110	10 (2,2,2,2,2)	f	s	-	*	-	-	(IY+d) & bit	
BIT b,r	11001011	01-b--r-			4 (2,2)	f		-	*	-	-	r & bit	
BOOL HL	11001100				2	fr		*	*	0	0	if (HL != 0) HL = 1	N
BOOL IX	11011101	11001100			4 (2,2)	f		*	*	0	0	if (IX != 0) IX = 1	N
BOOL IY	11111101	11001100			4 (2,2)	f		*	*	0	0	if (IY != 0) IY = 1	N
CALL mn	11001101	----n---	----m---		12 (2,2,2,3,3)			-	-	-	-	(SP-1) = PCH; (SP-2) = PCL; PC = mn; SP = SP-2	
CCF	00111111				2	f		-	-	-	*	CF = ~CF	
CP (HL)	10111110				5 (2,1,2)	f	s	*	*	V	*	A - (HL)	
CP (IX+d)	11011101	10111110	----d---		9 (2,2,2,1,2)	f	s	*	*	V	*	A - (IX+d)	
CP (IY+d)	11111101	10111110	----d---		9 (2,2,2,1,2)	f	s	*	*	V	*	A - (IY+d)	
CP n	11111110	----n---			4 (2,2)	f		*	*	V	*	A - n	
CP r	10111-r-				2	f		*	*	V	*	A - r	
CPL	00101111				2	r		-	-	-	-	A = ~A	
DEC (HL)	00110101				8 (2,1,2,3)	f	b	*	*	V	-	(HL) = (HL) - 1	
DEC (IX+d)	11011101	00110101	----d---		12 (2,2,2,1,2,3)	f	b	*	*	V	-	(IX+d) = (IX+d) -1	
DEC (IY+d)	11111101	00110101	----d---		12 (2,2,2,1,2,3)	f	b	*	*	V	-	(IY+d) = (IY+d) -1	
DEC IX	11011101	00101011			4 (2,2)			-	-	-	-	IX = IX - 1	
DEC IY	11111101	00101011			4 (2,2)			-	-	-	-	IY = IY - 1	
DEC r	00-r-101				2	fr		*	*	V	-	r = r - 1	
DEC ss	00s1				2	r		-	-	-	-	ss = ss - 1	
DJNZ e	00	--(e-2)-			5 (2,2,1)	r		-	-	-	-	B = B-1; if {B != 0} PC = PC + e	
EX (SP),HL	1110110				15 (2,2,1,2,2,3,3)	r		-	-	-	-	H <-> (SP+1); L <-> (SP)	M
EX (SP),IX	11110	111001			15 (2,2,1,2,2,3,3)			-	-	-	-	IXH <-> (SP+1); IXL <-> (SP)	
EX (SP),IY	1110	111001			15 (2,2,1,2,2,3,3)			-	-	-	-	IYH <-> (SP+1); IYL <-> (SP)	
EX AF,AF'	00				2			-	-	-	-	AF <-> AF'	
EX DE,HL	111011				2	s		-	-	-	-	if (!ALTD) then DE <-> HL else DE <-> HL'	N
EX DE',HL	111001				2	s		-	-	-	-	if (!ALTD) then DE' <-> HL else DE' <-> HL'	
EX	111001				2			-	-	-	-	BC <-> BC'; DE <-> DE'; HL <-> HL'	
INC (HL)	001				8 (2,1,2,3)	f	b	*	*	V	-	(HL) = (HL) + 1	
INC (IX+d)	1110	001	----d---		12 (2,2,2,1,2,3)	f	b	*	*	V	-	(IX+d) = (IX+d) + 1	
INC (IY+d)	1110	001	----d---		12 (2,2,2,1,2,3)	f	b	*	*	V	-	(IY+d) = (IY+d) + 1	
INC IX	1110	001			4 (2,2)			-	-	-	-	IX = IX + 1	
INC IY	1110	001			4 (2,2)			-	-	-	-	IY = IY + 1	
INC r	00-r-100				2	fr		*	*	V	-	r = r + 1	
INC ss	00ss0011				2	r		-	-	-	-	ss = ss + 1	
IOE	11011011				2			-	-	-	-	I/O external prefix	N
IOI	11010011				2			-	-	-	-	I/O internal prefix	N
IP 0	11101101	01000110			4 (2,2)			-	-	-	-	IP = {IP[5:0], 00}	NP
IP 1	11101101	01010110			4 (2,2)			-	-	-	-	IP = {IP[5:0], 01}	NP
IP 2	11101101	01001110			4 (2,2)			-	-	-	-	IP = {IP[5:0], 10}	NP
IP 3	11101101	01011110			4 (2,2)			-	-	-	-	IP = {IP[5:0], 11}	NP
IPRES	11101101	01011101			4 (2,2)			-	-	-	-	IP = {IP[1:0], IP[7:2]}	NP
JP (HL)	11101001				4 (2,2)			-	-	-	-	PC = HL	
JP (IX)	11011101	11101001			6 (2,2,2)			-	-	-	-	PC = IX	
JP (IY)	11111101	11101001			6 (2,2,2)			-	-	-	-	PC = IY	

JP f,mn	11-f-010	----n---	----m---		7 (2,2,2,1)			-	-	-	-	if {f} PC = mn	
JP mn	11000011	----n---	----m---		7 (2,2,2,1)			-	-	-	-	PC = mn	
JR cc,e	001cc000	--(e-2)-			5 (2,2,1)			-	-	-	-	if {cc} PC = PC + e	
JR e	00011000	--(e-2)-			5 (2,2,1)			-	-	-	-	PC = PC + e	
LCALL x,mn	11001111	----n---	----m---	---x----	19 (2,2,2,2,1,3,3,3,1)			-	-	-	-	(SP-1) = XPC; (SP-2) = PCH; (SP-3) = PCL; XPC = x; PC = mn; SP = SP-3	N
LD (BC),A	00000010				7 (2,2,3)		d	-	-	-	-	(BC) = A	
LD (DE),A	00010010				7 (2,2,3)		d	-	-	-	-	(DE) = A	
LD (HL),n	00110110	----n---			7 (2,2,3)		d	-	-	-	-	(HL) = n	
LD (HL),r	01110-r-				6 (2,1,3)		d	-	-	-	-	(HL) = r	
LD (HL+d),HL	11011101	11110100	----d---		13 (2,2,2,1,3,3)		d	-	-	-	-	(HL+d) = L; (HL+d+1) = H	N
LD (IX+d),HL	11110100	----d---			11 (2,2,1,3,3)		d	-	-	-	-	(IX+d) = L; (IX+d+1) = H	N
LD (IX+d),n	11011101	00110110	----d---	----n---	11 (2,2,2,2,3)		d	-	-	-	-	(IX+d) = n	
LD (IX+d),r	11011101	01110-r-	----d---		10 (2,2,2,1,3)		d	-	-	-	-	(IX+d) = r	
LD (IY+d),HL	11111101	11110100	----d---		13 (2,2,2,1,3,3)		d	-	-	-	-	(IY+d) = L; (IY+d+1) = H	N
LD (IY+d),n	11111101	00110110	----d---	----n---	11 (2,2,2,2,3)		d	-	-	-	-	(IY+d) = n	
LD (IY+d),r	11111101	01110-r-	----d---		10 (2,2,2,1,3)		d	-	-	-	-	(Iy+d) = r	
LD (mn),A	00110010	----n---	----m---		10 (2,2,2,1,3)		d	-	-	-	-	(mn) = A	
LD (mn),HL	00100010	----n---	----m---		13 (2,2,2,1,3,3)		d	-	-	-	-	(mn) = L; (mn+1) = H	
LD (mn),IX	11011101	00100010	----n---	----m---	15 (2,2,2,2,1,3,3)		d	-	-	-	-	(mn) = IXL; (mn+1) = IXH	
LD (mn),IY	11111101	00100010	----n---	----m---	15 (2,2,2,2,1,3,3)		d	-	-	-	-	(mn) = IYL; (mn+1) = IYH	
LD (mn),ss	11101101	01ss0011	----n---	----m---	15 (2,2,2,2,1,3,3)		d	-	-	-	-	(mn) = ssl; (mn+1) = ssh	
LD (SP+n),HL	11010100	----n---			11 (2,2,1,3,3)			-	-	-	-	(SP+n) = L; (SP+n+1) = H	N
LD (SP+n),IX	11011101	11010100	----n---		13 (2,2,2,1,3,3)			-	-	-	-	(SP+n) = IXL; (SP+n+1) = IXH	N
LD (SP+n),IY	11111101	11010100	----n---		13 (2,2,2,1,3,3)			-	-	-	-	(SP+n) = IYL; (SP+n+1) = IYH	N
LD A,(BC)	00001010				6 (2,2,2)	r	s	-	-	-	-	A = (BC)	
LD A,(DE)	00011010				6 (2,2,2)	r	s	-	-	-	-	A = (DE)	
LD A,(mn)	00111010	----n---	----m---		9 (2,2,2,1,2)	r	s	-	-	-	-	A = (mn)	
LD A,EIR	11101101	01010111			4 (2,2)	fr		*	*	-	-	A = EIR	
LD A,IIR	11101101	01011111			4 (2,2)	fr		*	*	-	-	A = IIR	
LD A,XPC	11101101	01110111			4 (2,2)	r		-	-	-	-	A = XPC	N
LD dd,(mn)	11101101	01dd1011	----n---	----m---	13 (2,2,2,2,1,2,2)	r	s	-	-	-	-	ddl = (mn); ddh = (mn+1)	
LD dd',BC	11101101	01dd1001			4 (2,2)			-	-	-	-	dd' = BC (dd': 00-BC', 01-DE', 10-HL')	N
LD dd',DE	11101101	01dd0001			4 (2,2)			-	-	-	-	dd' = DE (dd': 00-BC', 01-DE', 10-HL')	N
LD dd,mn	00dd0001	----n---	----m---		6 (2,2,2)	r		-	-	-	-	dd = mn	
LD EIR,A	11101101	01000111			4 (2,2)				-	-	-	EIR = A	
LD IIR,A	11101101	01001111			4 (2,2)				-	-	-	IIR = A	
LD HL,(mn)	00101010				11 (2,2,2,1,2,2)		s		-	-	-	L = (mn); H = (mn+1)	
LD HL,(HL+d)	11011101	11100100	----d---		11 (2,2,2,1,2,2)		s		-	-	-	L = (HL+d); H = (HL+d+1)	
LD HL,(IY+d)	11111101	11100100	----d---		11 (2,2,2,1,2,2)		s		-	-	-	L = (IY+d); H = (IY+d+1)	
LD HL,(SP+n)	11000100				9 (2,2,1,2,2)				-	-	-	L = (SP+n); H = (SP+n+1)	
		01111100			4 (2,2)				-	-	-	HL = IX	
	11111101	01111100			4 (2,2)				-	-	-	HL = IY	
			----n---						-	-	-		
		01111101			4 (2,2)					-	-	IX = HL	

LD IX,mn	11011101	00100001	----n---	----m---	8 (2,2,2,2)			-	-	-	-	IX = mn	
LD IY,(mn)	11111101	00101010	----n---	----m---	13 (2,2,2,2,1,2,2)		s	-	-	-	-	IYL = (mn); IYH = (mn+1)	
LD IY,(SP+n)	11111101	11000100	----n---		11 (2,2,2,1,2,2)			-	-	-	-	IYL = (SP+n); IYH = (SP+n+1)	N
LD IY,HL	11111101	01111101			4 (2,2)			-	-	-	-	IY = HL	N
LD IY,mn	11111101	00100001	----n---	----m---	8 (2,2,2,2)			-	-	-	-	IY = mn	
LD r,(HL)	01-r-110				5 (2,1,2)	r	s	-	-	-	-	r = (HL)	
LD r,(IX+d)	11011101	01-r-110	----d---		9 (2,2,2,1,2)	r	s	-	-	-	-	r = (IX+d)	
LD r,(IY+d)	11111101	01-r-110	----d---		9 (2,2,2,1,2)	r	s	-	-	-	-	r = (IY+d)	
LD XPC,A	11101101	01100111			4 (2,2)			-	-	-	-	XPC = A	NP
LD r,n	00-r-110	----n---			4 (2,2)	r		-	-	-	-	r = n	
LD r,g	01-r--r'				2	r		-	-	-	-	r = r'	
LD SP,HL	11111001				2			-	-	-	-	SP = HL	P
LD SP,IX	11011101	11111001			4 (2,2)			-	-	-	-	SP = IX	P
LD SP,IY	11111101	11111001			4 (2,2)			-	-	-	-	SP = IY	P
LDD	11101101	10101000			10 (2,2,1,2,3)		d	-	-	*	-	(DE) = (HL); BC = BC-1; DE = DE-1; HL = HL-1	
LDDR	11101101	10111000			6+7i (2,2,1,(2,3,2)i,1)		d	-	-	*	-	if {BC != 0} repeat: (DE) 0 Tc ( )Tj. f 553.68 7.92 1 Tf .sc (-)19f 553	

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Clock cycles	A	I	S	Z	LV	C	Operation	N/M/P
OR HL,DE	11101100				2	fr		*	*	L	0	HL = HL | DE	N
OR IX,DE	11011101	11101100			4 (2,2)	f		*	*	L	0	IX = IX | DE	N
OR IY,DE	11111101	11101100			4 (2,2)	f		*	*	L	0	IY = IY | DE	N
OR n	11110110	----n---			4 (2,2)	fr		*	*	L	0	A = A | n	
OR r	10110-r-				2	fr		*	*	L	0	A = A | r	
POP IP	11101101	01111110			7 (2,2,1,2)			-	-	-	-	IP = (SP); SP = SP+1	NP
POP IX	11011101	11100001			9 (2,2,1,2,2)			-	-	-	-	IXL = (SP); IXH = (SP+1); SP = SP+2	
POP IY	11111101	11100001			9 (2,2,1,2,2)			-	-	-	-	IYL = (SP); IYH = (SP+1); SP = SP+2	
POP zz	11zz0001				7 (2,1,2,2)	r		-	-	-	-	zzl = (SP); zzh = (SP+1); SP = SP+2	
PUSH IP	11101101	01110110			9 (2,2,2,3)			-	-	-	-	(SP-1) = IP; SP = SP-1	N
PUSH IX	11011101	11100101			12 (2,2,2,3,3)			-	-	-	-	(SP-1) = IXH; (SP-2) = IXL; SP = SP-2	
PUSH IY	11111101	11100101			12 (2,2,2,3,3)			-	-	-	-	(SP-1) = IYH; (SP-2) = IYL; SP = SP-2	
PUSH zz	11zz0101				10 (2,2,3,3)			-	-	-	-	(SP-1) = zzh; (SP-2) = zzl; SP = SP-2	
RES b,(HL)	11001011	10-b-110			10 (2,2,1,2,3)		d	-	-	-	-	(HL) = (HL) & ~bit	
RES b,(IX+d)	11011101	11001011	----d---	10-b-110	13 (2,2,2,2,2,3)		d	-	-	-	-	(IX+d) = (IX+d) & ~bit	
RES b,(IY+d)	11111101	11001011	----d---	10-b-110	13 (2,2,2,2,2,3)		d	-	-	-	-	(IY+d) = (IY+d) & ~bit	
RES b,r	11001011	10-b--r-			4 (2,2)	r		-	-	-	-	r = r & ~bit	
RET	11001001				8 (2,1,2,2,1)			-	-	-	-	PCL = (SP); PCH = (SP+1); SP = SP+2	
RET f	11-f-000				2 8 (2,1,2,2,1)			-	-	-	-	if {f} PCL = (SP); PCH = (SP+1); SP = SP+2	
RETI	11101101	01001101			12 (2,2,1,2,2,2,1)			-	-	-	-	IP = (SP); PCL = (SP+1); PCH = (SP+2); SP = SP+3	NP
RL (HL)	11001011	00010110			10 (2,2,1,2,3)	f	b	*	*	L	*	{CY,(HL)} = {(HL),CY}	
RL (IX+d)	11011101	11001011	----d---	00010110	13 (2,2,2,2,2,3)	f	b	*	*	L	*	{CY,(IX+d)} = {(IX+d),CY}	
RL (IY+d)	11111101	11001011	----d---	00010110	13 (2,2,2,2,2,3)	f	b	*	*	L	*	{CY,(IY+d)} = {(IY+d),CY}	
RL DE	11110011				2	fr		*	*	L	*	{CY,DE} = {DE,CY}	N
RL r	11001011	00010-r-			4 (2,2)	fr		*	*	L	*	{CY,r} = {r,CY}	
RLA	00010111				2	fr		-	-	-	*	{CY,A} = {A,CY}	
RLC (HL)	11001011	00000110			10 (2,2,1,2,3)	f	b	*	*	L	*	(HL) = {(HL)[6,0],(HL)[7]}; CY = (HL)[7]	
RLC (IX+d)	11011101	11001011	----d---	00000110	13 (2,2,2,2,2,3)	f	b	*	*	L	*	(IX+d) = {(IX+d)[6,0],(IX+d)[7]}; CY = (IX+d)[7]	
RLC (IY+d)	11111101	11001011	----d---	00000110	13 (2,2,2,2,2,3)	f	b	*	*	L	*	(IY+d) = {(IY+d)[6,0],(IY+d)[7]}; CY = (IY+d)[7]	
RLC r	11001011	00000-r-			4 (2,2)	fr		*	*	L	*	r = {r[6,0],r[7]}; CY = r[7]	
RLCA	00000111				2	fr		-	-	-	*	A = {A[6,0],A[7]}; CY = A[7]	
RR (HL)	11001011	00011110			10 (2,2,1,2,3)	f	b	*	*	L	*	{(HL),CY} = {CY,(HL)}	
RR (IX+d)	11011101	11001011	----d---	00011110	13 (2,2,2,2,2,3)	f	b	*	*	L	*	{(IX+d),CY} = {CY,(IX+d)}	
RR (IY+d)	11111101	11001011	----d---	00011110	13 (2,2,2,2,2,3)	f	b	*	*	L	*	{(IY+d),CY} = {CY,(IY+d)}	
RR DE	11111011				2	fr		*	*	L	*	{DE,CY} = {CY,DE}	N
RR HL	11111100				2	fr		*	*	L	*	{HL,CY} = {CY,HL}	N
RR IX	11011101	11111100			4 (2,2)	f		*	*	L	*	{IX,CY} = {CY,IX}	N
RR IY	11111101	11111100			4 (2,2)	f		*	*	L	*	{IY,CY} = {CY,IY}	N
RR r	11001011	00011-r-			4 (2,2)	fr		*	*	L	*	{r,CY} = {CY,r}	
RRA	00011111				2	fr		-	-	-	*	{A,CY} = {CY,A}	
RRC (HL)	11001011	00001110			10 (2,2,1,2,3)	f	b	*	*	L	*	(HL) = {(HL)[0],(HL)[7,1]}; CY = (HL)[0]	
RRC (IX+d)	11011101	11001011	----d---	00001110	13 (2,2,2,2,2,3)	f	b	*	*	L	*	(IX+d) = {(IX+d)[0],(IX+d)[7,1]}; CY = (IX+d)[0]	
RRC (IY+d)	11111101	11001011	----d---	00001110	13 (2,2,2,2,2,3)	f	b	*	*	L	*	(IY+d) = {(IY+d)[0],(IY+d)[7,1]}; CY = (IY+d)[0]	
RRC r	11001011	00001-r-			4 (2,2)	fr		*	*	L	*	r = {r[0],r[7,1]}; CY = r[0]	

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Clock cycles	A	I	S	Z	LV	C	Operation	N/M/P
RRCA	00001111				2	fr		-	-	-	*	A = {A[0],A[7,1]}; CY = A[0]	
RST v	11-v-111				8 (2,2,2,2)			-	-	-	-	(SP-1) = PCH; (SP-2) = PCL; SP = SP - 2; PC = {R, 0, v, 0000}	
SBC (IX+d)	11011101	10011110	----d---		9 (2,2,2,1,2)	fr	s	*	*	V	*	A = A - (IX+d) - CY	
SBC (IY+d)	11111101	10011110	----d---		9 (2,2,2,1,2)	fr	s	*	*	V	*	A = A - (IY+d) - CY	
SBC A,(HL)	10011110				5 (2,1,2)	fr	s	*	*	V	*	A = A - (HL) - CY	
SBC A,n	11011110	----n---			4 (2,2)	fr		*	*	V	*	A = A - n - CY	
SBC A,r	10011-r-				2	fr		*	*	V	*	A = A - r - CY	
SBC HL,ss	11101101	01ss0010			4 (2,2)	fr		*	*	V	*	HL = HL - ss - CF	
SCF	00110111				2	f		-	-	-	1	CF = 1	
SET b,(HL)	11001011	11-b-110			10 (2,2,1,2,3)		b	-	-	-	-	(HL) = (HL) | bit	
SET b,(IX+d)	11011101	11001011	----d---	11-b-110	13 (2,2,2,2,2,3)		b	-	-	-	-	(IX+d) = (IX+d) | bit	
SET b,(IY+d)	11111101	11001011	----d---	11-b-110	13 (2,2,2,2,2,3)		b	-	-	-	-	(IY+d) = (IY+d) | bit	
SET b,r	11001011	11-b--r-			4 (2,2)	r		-	-	-	-	r = r | bit	
SLA (HL)	11001011	00100110			10 (2,2,1,2,3)	f	b	*	*	L	*	(HL) = {(HL)[6,0],0}; CY = (HL)[7]	
SLA (IX+d)	110111	11001011	----d---	00100110	13 (2,2,2,2,2,3)	f	b	*	*	L	*	(IX+d) = {(IX+d)[6,0],0}; CY = (IX+d)[7]	
SLA (IY+d)	1111111	11001011	----d---	00100110	13 (2,2,2,2,2,3)	f	b	*	*	L	*	(IY+d) = {(IY+d)[6,0],0}; CY = (IY+d)[7]	
SLA r	11001011	00100-r-			4 (2,2)	fr		*	*	L	*	r = {r[6,0],0}; CY = r[7]	
SRA (HL)	11001011	0010111			10 (2,2,1,2,3)	f	b	*	*	L	*	(HL) = {(HL)[7],(HL)[7,1]}; CY = (HL)[0]	
SRA (IX+d)	11110	11001011	----d---	0010111	13 (2,2,2,2,2,3)	f	b	*	*	L	*	(IX+d) = {(IX+d)[7],(IX+d)[7,1]}; CY = (IX+d)[0]	
SRA (IY+d)	1111110	11001011	----d---	0010111	13 (2,2,2,2,2,3)	f	b	*	*	L	*	(IY+d) = {(IY+d)[7],(IY+d)[7,1]}; CY = (IY+d)[0]	
SRA r	11001011	00101-r-			4 (2,2)	fr		*	*	L	*	r = {r[7],r[7,1]}; CY = r[0]	
SRL (HL)	11001011	0011111			10 (2,2,1,2,3)	f	b	*	*	L	*	(HL) = {0,(HL)[7,1]}; CY = (HL)[0]	
SRL (IX+d)	111110	11001011	----d---	0011111	13 (2,2,2,2,2,3)	f	b	*	*	L	*	(IX+d) = {0,(IX+d)[7,1]}; CY = (IX+d)[]	
SRL (IY+d)	1111110	11001011	----d---	0011111	13 (2,2,2,2,2,3)	f	b	*	*	L	*	(IY+d) = {0,(IY+d)[7,1]}; CY = (IY+d)[]	
SRL r	11001011	00111-r-			4 (2,2)	fr		*	*	L	*	r = {,r[7,1]}; CY = r[0]	
SUB (HL)	10010110				5 (2,1,2)	fr	s	*	*	V	*	A = A - (HL)	
SUB (IX+d)	11011101	10010110	----d---		9 (2,2,2,1,2)	fr	s	*	*	V	*	A = A - (IX+d)	
SUB (IY+d)	11111101	10010110	----d---		9 (2,2,2,1,2)	fr	s	*	*	V	*	A = A - (IY+d)	
SUB n	11010110	----n---			4 (2,2)	fr		*	*	V	*	A = A - n	
SUB r	10010-r-				2	fr		*	*	V	*	A = A - r	
XOR (HL)	10101110				5 (2,1,2)	fr	s	*	*	L	0	A = [A & ~(HL)] | [~A & (HL)]	
XOR (IX+d)	11011101	10101110	----d---		9 (2,2,2,1,2)	fr	s	*	*	L	0	A = [A & ~(IX+d)] | [~A & (IX+d)]	
XOR (IY+d)	11111101	10101110	----d---		9 (2,2,2,1,2)	fr	s	*	*	L	0	A = [A & ~(IY+d)] | [~A & (IY+d)]	
XOR n	11101110	----n---			4 (2,2)	fr		*	*	L	0	A = [A & ~n] | [~A & n]	
XOR r	10101-r-				2	fr		*	*	L	0	A = [A & ~r] | [~A & r]	

*