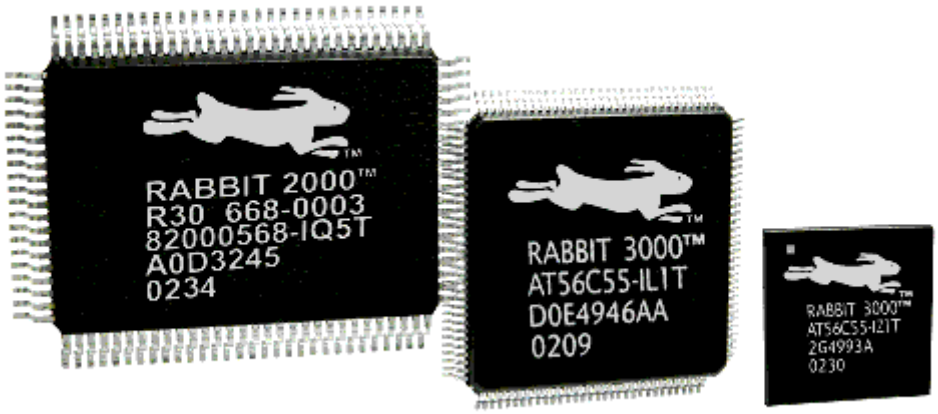
Rabbit 2000/3000 Microprocessor

Instruction Reference Manual

019–0098 F • 040114



This manual (or an even more up-to-date revision) is available for free download at the Rabbit website: www.rabbitsemiconductor.com

Table of Contents

1. Alphabetical Listing of Instructions 1
2. Instructions Listed by Group 3
3. Document Conventions 7
4. Processor Registers 11
5. OpCode Descriptions 13
6. Opcode Map 153
7. Quick Reference Table 161

Notice to Users 169

1. Alphabetical Listing of Instructions

A

ADC A,n 14
ADC A,r 15
ADC A,(HL) 13
ADC A,(IX+d) 13
ADC A,(IY+d) 13
ADC HL,ss 16
ADD A,n 18
ADD A,r 19
ADD A,(HL) 17
ADD A,(IX+d) 17
ADD A,(IY+d) 17
ADD HL,ss 20
ADD IX,xx 21
ADD IY,yy 21
ADD SP,d 22
ALTD 23
AND HL,DE 25
AND IX,DE 25
AND IY,DE 25
AND n 26
AND r 27
AND (HL) 24
AND (IX+d) 24
AND (IY+d) 24

B

BIT b,r 29
BIT b,(HL) 28
BIT b,(IX+d) 28
BIT b,(IY+d) 28
BOOL HL 30
BOOL IX 31
BOOL IY 31

C

CALL mn 32
CCF 33
CP n 35
CP r 36
CP (HL) 34
CP (IX+d) 34
CP (IY+d) 34
CPL 37

D

DEC IX 39
DEC IY 39
DEC r 40
DEC ss 41
DEC (HL) 38
DEC (IX+d) 38
DEC (IY+d) 38
DJNZ e 42

E

EX AF,AF' 45
EX DE',HL 46
EX DE,HL 46
EX (SP),HL 43
EX (SP),IX 44
EX (SP),IY 44
EXX 47

I

IDET 48
INC IX 50
INC IY 50
INC r 51
INC ss 52
INC (HL) 49
INC (IX+d) 49
INC (IY+d) 49
IOE 53
IOI 53
IPRES 55
IPSET 0 54
IPSET 1 54
IPSET 2 54
IPSET 3 54

J

JP f,mn 57
JP mn 56
JP (HL) 56
JP (IX) 56
JP (IY) 56
JR cc,e 58
JR e 59

L

LCALL x,mn 60
LD A,EIR 68
LD A,IIR 68
LD A,XPC 69
LD A,(BC) 67
LD A,(DE) 67
LD A,(mn) 67
LD dd',BC 71
LD dd',DE 71
LD dd,mn 72
LD dd,(mn) 70
LD EIR,A 73
LD HL,IX 76
LD HL,IY 76
LD HL,(HL+d) 74
LD HL,(IX+d) 74
LD HL,(IY+d) 74
LD HL,(mn) 74
LD HL,(SP+n) 75
LD IIR,A 73
LD IX,HL 79
LD IX,mn 79
LD IX,(mn) 77
LD IX,(SP+n) 78
LD IY,HL 79
LD IY,mn 79
LD IY,(mn) 80
LD IY,(SP+n) 81
LD r,g 84
LD r,n 83
LD r,(HL) 82
LD r,(IX+d) 82
LD r,(IY+d) 82
LD SP,HL 85
LD SP,IX 85
LD SP,IY 85
LD XPC,A 86
LD (BC),A 61
LD (DE),A 61
LD (HL),n 61
LD (HL),r 61
LD (HL+d),HL 62
LD (IX+d),HL 63
LD (IX+d),n 63
LD (IX+d),r 63
LD (IY+d),HL 64

LD (IY+d),n 64
LD (IY+d),r 64
LD (mn),A 65
LD (mn),HL 65
LD (mn),IX 65
LD (mn),IY 65
LD (mn),ss 65
LD (SP+n),HL 66
LD (SP+n),IX 66
LD (SP+n),IY 66
LDD 87
LDDR 87
LDDSR 88
LDI 87
LDIR 87
LDISR 88
LDP HL,(HL) 91
LDP HL,(IX) 91
LDP HL,(IY) 91
LDP HL,(mn) 92
LDP IX,(mn) 92
LDP IY,(mn) 92
LDP (HL),HL 89
LDP (IX),HL 89
LDP (IY),HL 89
LDP (mn),HL 90
LDP (mn),IX 90
LDP (mn),IY 90
LJP x,mn 93
LRET 94
LSDDR 95
LSDR 95
LSIDR 95
LSIR 95

M

MUL 96

N

NEG 97
NOP 98

O

OR HL,DE 100
OR IX,DE 101
OR IY,DE 101
OR n 102
OR r 102
OR (HL) 99
OR (IX+d) 99
OR (IY+d) 99

P

POP IP 103
POP IX 103
POP IY 103
POP SU 104
POP zz 105
PUSH IP 106
PUSH IX 106
PUSH IY 106
PUSH SU 107
PUSH zz 108

R

RA 126
RDMODE 109
RES b,r 111
RES b,(HL) 110
RES b,(IX+d) 110
RES b,(IY+d) 110
RET 112
RET f 113
RETI 114
RL DE 116
RL r 117
RL (HL) 115
RL (IX+d) 115
RL (IY+d) 115
RLA 118
RLC r 120
RLC (HL) 119
RLC (IX+d) 119
RLC (IY+d) 119
RLCA 121
RR DE 123
RR HL 123
RR IX 124
RR IY 124
RR r 125
RR (HL) 122
RR (IX+d) 122
RR (IY+d) 122
RRC r 128
RRC (HL) 127
RRC (IX+d) 127
RRC (IY+d) 127
RRCA 129
RST v 130

S

SBC A,n 132
SBC A,r 132
SBC A,(HL) 131
SBC HL,ss 133
SBC (IX+d) 131
SBC (IY+d) 131
SCF 134
SET b,r 136
SET b,(HL) 135
SET b,(IX+d) 135
SET b,(IY+d) 135
SETUSR 137
SLA r 139
SLA (HL) 138
SLA (IX+d) 138
SLA (IY+d) 138
SRA r 141
SRA (HL) 140
SRA (IX+d) 140
SRA (IY+d) 140
SRL r 143
SRL (HL) 142
SRL (IX+d) 142
SRL (IY+d) 142
SUB n 145
SUB r 146
SUB (HL) 144
SUB (IX+d) 144
SUB (IY+d) 144
SURES 147
SYSCALL 148

U

UMA 149
UMS 149

X

XOR n 151
XOR r 152
XOR (HL) 150
XOR (IX+d) 150
XOR (IY+d) 150

2. Instructions Listed by Group

A. Load Immediate Data

LD dd,mn 72
LD IX,mn 79
LD IY,mn 79
LD r,n 83

B. Load and Store to an Immediate Address

LD (mn),A 65
LD (mn),HL 65
LD (mn),IX 65
LD (mn),IY 65
LD (mn),ss 65
LD A,(mn) 67
LD dd,(mn) 70
LD HL,(mn) 74
LD IX,(mn) 77
LD IY,(mn) 80

C. 8-bit Indexed Load and Store

LD (BC),A 61
LD (DE),A 61
LD (HL),n 61
LD (HL),r 61
LD (IX+d),n 63
LD (IX+d),r 63
LD (IY+d),n 64
LD (IY+d),r 64
LD A,(BC) 67
LD A,(DE) 67
LD r,(HL) 82
LD r,(IX+d) 82
LD r,(IY+d) 82

D. 16-bit Indexed Load and Store

LD (HL+d),HL 62
LD (IX+d),HL 63
LD (IY+d),HL 64
LD (SP+n),HL 66
LD (SP+n),IX 66
LD (SP+n),IY 66
LD HL,(HL+d) 74
LD HL,(IX+d) 74
LD HL,(IY+d) 74
LD HL,(SP+n) 75
LD IX,(SP+n) 78
LD IY,(SP+n) 81

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

LDP (HL),HL 89
LDP (IX),HL 89
LDP (IY),HL 89
LDP (mn),HL 90
LDP (mn),IX 90
LDP (mn),IY 90
LDP HL,(HL) 91
LDP HL,(IX) 91
LDP HL,(IY) 91
LDP HL,(mn) 92
LDP IX,(mn) 92
LDP IY,(mn) 92

F. Register to Register Moves

LD A,EIR 68
LD A,IIR 68
LD A,XPC 69
LD dd',BC 71
LD dd',DE 71
LD EIR,A 73
LD HL,IX 76
LD HL,IY 76
LD IIR,A 73
LD IX,HL 79
LD IY,HL 79
LD r,g 84
LD SP,HL 85
LD SP,IX 85
LD SP,IY 85
LD XPC,A 86

G. Exchange

EX (SP),HL 43
EX (SP),IX 44
EX (SP),IY 44
EX AF,AF' 45
EX DE,HL 46
EX DE',HL 46

EXX 47

H. Stack Manipulation

ADD SP,d 22
POP IP 103
POP IX 103
POP IY 103
POP zz 105
PUSH IP 106
PUSH IX 106
PUSH IY 106
PUSH zz 108

I. 16-bit Arithmetic, Logical, and Rotate

ADC HL,ss 16
ADD HL,ss 20
ADD IX,xx 21
ADD IY,yy 21
ADD SP,d 22
AND HL,DE 25
BOOL HL 30
BOOL IX 31
BOOL IY 31
DEC IX 39
DEC IY 39
DEC ss 41
INC IX 50
INC IY 50
INC ss 52
MUL 96
NEG 97
OR HL,DE 100
OR IX,DE 101
OR IY,DE 101
RL DE 116
RR DE 123
RR HL 123
RR IX 124
RR IY 124
SBC HL,ss 133

I.16-bit Arithmetic, Logical, and Rotate

AND HL,DE 25
AND IX,DE 25
AND IY,DE 25

J. 8-bit Arithmetic and Logical

ADC A,(HL) 13
ADC A,(IX+d) 13
ADC A,(IY+d) 13
ADC A,n 14
ADC A,r 15
ADD A,(HL) 17
ADD A,(IX+d) 17
ADD A,(IY+d) 17
ADD A,n 18
ADD A,r 19
AND (HL) 24
AND (IX+d) 24
AND (IY+d) 24
AND r 27
CP (HL) 34
CP (IX+d) 34
CP (IY+d) 34
CP n 35
CP r 36
NEG 97
OR (HL) 99
OR (IX+d) 99
OR (IY+d) 99
OR n 102
OR r 102
SBC (IX+d) 131
SBC (IY+d) 131
SBC A,(HL) 131
SBC A,n 132
SBC A,r 132
SUB (HL) 144
SUB (IX+d) 144
SUB (IY+d) 144
SUB n 145
SUB r 146
XOR (HL) 150
XOR (IX+d) 150
XOR (IY+d) 150
XOR n 151
XOR r 152

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

BIT b,(HL) 28
BIT b,(IX+d) 28
BIT b,(IY+d) 28

BIT b,r 29
RES b,(HL) 110
RES b,(IX+d) 110
RES b,(IY+d) 110
RES b,r 111
SET b,(HL) 135
SET b,(IX+d) 135
SET b,(IY+d) 135
SET b,r 136

L. 8-bit Increment and Decrement

DEC (HL) 38
DEC (IX+d) 38
DEC (IY+d) 38
DEC r 40
INC (HL) 49
INC (IX+d) 49
INC (IY+d) 49
INC r 51

M. 8-bit Fast Accumulator

CPL 37
RLA 118
RLCA 121
RRA 126
RRCA 129

N. 8-bit Shift and Rotate

RL (HL) 115
RL (IX+d) 115
RL (IY+d) 115
RLC (HL) 119
RLC (IX+d) 119
RLC (IY+d) 119
RLC r 120
RR (HL) 122
RR (IX+d) 122
RR (IY+d) 122
RR r 125
RRC (HL) 127
RRC (IX+d) 127
RRC (IY+d) 127
RRC r 128
SLA (HL) 138
SLA (IX+d) 138
SLA (IY+d) 138
SLA r 139
SRA (HL) 140
SRA (IX+d) 140
SRA (IY+d) 140
SRA r 141
SRL (HL) 142
SRL (IX+d) 142
SRL (IY+d) 142
SRL r 143

O. Instruction Prefixes

ALTD 23
IOE 53
IOI 53

P. Block Moves

LDD 87
LDDR 87
LDDSR 88
LDI 87
LDIR 87
LDISR 88
LSDDR 95
LSDR 95
LSIDR 95
LSIR 95

Q. Control, Jump, and Call

CALL mn 32
DJNZ e 42
JP (HL) 56
JP (IX) 56
JP (IY) 56
JP f,mn 57
JP mn 56
JR cc,e 58
JR e 59
LCALL x,mn 60
LJP x,mn 93
LRET 94
RET 112
RET f 113
RETI 114
RST v 130

R. Miscellaneous

CCF 33
IPSET 0 54
IPSET 1 54
IPSET 2 54
IPSET 3 54
NOP 98
SCF 134

S. Special Arithmetic

UMA 149
UMS 149

T. Privileged Instructions

BIT b,(HL) 28
IPRES 55
IPSET 0 54
IPSET 1 54
IPSET 2 54
IPSET 3 54
LD A,XPC 69
LD SP,HL 85
LD SP,IX 85
LD SP,IY 85
LD XPC,A 86
POP IP 103
RETI 114

U. Rabbit 3000A Instructions

IDET 48
LDDSR 88
LDISR 88
LSDDR 95
LSDR 95
LSIDR 95
LSIR 95
POP SU 104
PUSH SU 107
RDMODE 109
SETUSR 137
SURES 147
SYSCALL 148
UMA 149
UMS 149

W. System/User Mode

IDET 48
POP SU 104
PUSH SU 107
RDMODE 109
SETUSR 137
SURES 147
SYSCALL 148

3. Document Conventions

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: Typical Clocks Breakdown

Process	Clocks
Each byte of the opcode.	2
Each data byte read.	2
Write to memory or external IO.	3
Write to internal IO.	2
Internal operation or computation.	1

- **`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

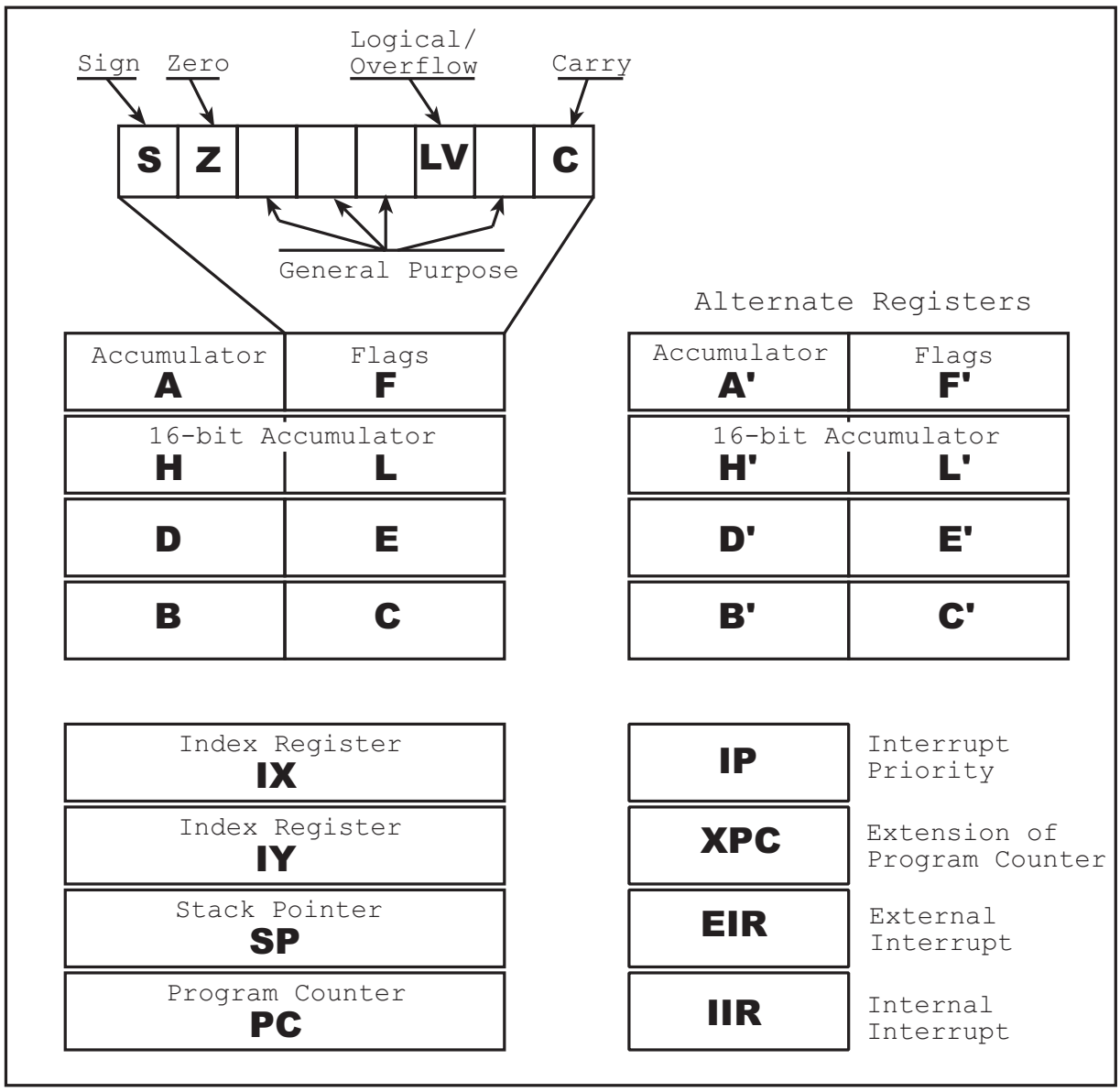
Rabbit	Z180	Meaning
b	*b*	Bit select (000 = bit 0, 001 = bit 1, 010 = bit 2, 011 = bit 3, 100 = bit 4, 101 = bit 5, 110 = bit 6, 111 = bit 7)
cc	*cc*	Condition code select (00 = NZ, 01 = Z, 10 = NC, 11 = C)
d	*d*	7-bit (signed) displacement. Expressed in two's complement.
dd	*ww*	word register select-destination (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, g	*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 Zero
Z	Z=1	Zero
NC	C=0	No Carry (C=0)
C	C=1	Carry (C=1)
P	S=0	Positive
M	S=1	Minus
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

4. Processor Registers



5. 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 A is summed with the C flag and with the data in memory whose location is:

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

The result is then stored in A.

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 C flag and with the data in A. The sum is then stored in A.

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

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 HL is summed with the C flag and with the data in *ss* (any of 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 A is summed with the data in the memory location whose address is:

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

The result is stored in A.

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

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

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

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

ADD SP,*d*

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 (SP) is summed with the 7-bit signed displacement *d*, and then stored in SP.

ALTD

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

The instructions

```
ALTD LD DE,BC
```

and

```
LD DE',BC
```

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

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

Opcode	Instruction	Clocks	Operation
A6	AND (HL)	5 (2,1,2)	A = A & (HL)
DD A6 *d*	AND (IX+*d*)	9 (2,2,2,1,2)	A = A & (IX+*d*)
FD A6 *d*	AND (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 AND operation between the byte in A and the byte whose address is:

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

The relative bits of each byte are compared (i.e., bit 0 of both bytes are compared, bit 1 of both bytes are compared, etc.). The associated bit in the result byte is set only if both the compared bits are set. The result is stored in A.

Example

If the byte in A contains the value 1011 1100 and the byte at memory location HL contains the value 1101 0101, then the execution of the instruction:

```
AND (HL)
```

would result in the byte in A becoming 1001 0100.

AND HL,DE

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 HL and the word in DE. The relative bits of each byte are compared (i.e., bit 0 of both bytes are compared, bit 1 of both bytes are compared, etc.). 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
AND IY,DE

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

The relative bits of each byte are compared (i.e., bit 0 of both bytes are compared, bit 1 of both bytes are compared, etc.). 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 A and the 8-bit constant *n*. The relative bits of each byte are compared (i.e., bit 0 of both bytes are compared, bit 1 of both bytes are compared, etc.). The associated bit in the result byte is set only if both the compared bits are set. The result is stored in A.

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 A and the byte in *r* (any of the registers A, B, C, D, E, H, or L). The relative bits of each byte are compared (i.e., bit 0 of both bytes are compared, bit 1 of both bytes are compared, etc.). The associated bit in the result byte is set only if both the compared bits are set. The result is stored in A.

BIT *b*,(HL)
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 HL, or
- the sum of data in IX plus a displacement value *d*, or
- the sum of data in IY plus a displacement value *d*.

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

`BIT b,(HL)` is a privileged instruction.

BIT *b,r*

Opcode								Instruction	Clocks	Operation
b,r	*A*	*B*	*C*	*D*	*E*	*H*	*L*	BIT b,r	4(2,2)	r & bit
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 bit *b* (any of the bits 0, 1, 2, 3, 4, 5, 6, or 7) of the byte in *r* (any of the registers A, B, C, D, E, H, or L). The Z flag is set if the tested bit is 0, reset if the bit is 1.

BOOL HL

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 HL does not equal zero, then it is set to 1.

BOOL IX
BOOL IY

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 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 PC is pushed onto the stack. The high-order byte of PC is pushed first, then the low-order byte. PC is then loaded with *mn*,16-bit address of the first instruction of the subroutine. SP 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 C 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 A with the data whose address is contained in:

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

These compares are accomplished by subtracting the appropriate data ((HL), (IX+*d*), or (IY+*d*)) from A. The result is:

```
A < x : S=1, C=1, Z=0, L/V=V

A = x : S=0, C=0, Z=1, L/V=V

A > x : S=0, C=0, Z=0, L/V=V
```

Where x is (HL), (IX+d), or (IY+d) and “V” indicates that the overflow flag is set on an arithmetic overflow result. That is, the overflow flag is set when the operands have different signs and the sign of the result is different from the argument you are subtracting from (A in this case). For example, if A contains 0x80 and you're comparing it to 0x01 the overflow flag will be set.

This operation does not affect the data in A.

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 A with an 8-bit constant *n*. This compare is accomplished by subtracting *n* from A. The result is:

```
A < n : S=1, C=1, Z=0, L/V=V

A = n : S=0, C=0, Z=1, L/V=V

A > n : S=0, C=0, Z=0, L/V=V
```

“V” indicates that the overflow flag is set on an arithmetic overflow result. That is, the overflow flag is signalled when the operands have different signs and the sign of the result is different from the argument you are subtracting from (A in this case). For example if A contains 0x80 and you're comparing it to 0x01 the overflow flag will be set.

This operation does not affect the data in A.

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 A 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 data in register *r* from A. The result is:

```
A < x : S=1, C=1, Z=0, L/V=V

A = x : S=0, C=0, Z=1, L/V=V

A > x : S=0, C=0, Z=0, L/V=V
```

Where “x” is the data in register *r* and “V” indicates that the overflow flag is set on an arithmetic overflow result. That is, the overflow flag is signalled when the operands have different signs and the sign of the result is different from the argument you are subtracting from (A in this case). For example if A contains 0x80 and you're comparing it to 0x01 the overflow flag will be set.

This operation does not affect the data in A.

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 A is inverted (one’s complement).

Example

If the data in A is 1100 0101, after the instruction CPL A 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 HL, or
- the data in IX plus a displacement value *d*, or
- the data in 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 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 *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**
0B	DEC BC	2	BC = BC - 1
1B	DEC DE	2	DE = DE - 1
2B	DEC HL	2	HL = HL - 1
3B	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 *ss* (any of 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 B then, if the data in B does not equal 0, it adds the 8 bit signed constant *e* to PC.

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

EX (SP),HL

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 H with the data whose address is the data in SP plus 1; and exchanges the byte in L with the data whose address is the data in SP.

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

EX DE,HL
EX DE',HL

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 DE with the data in HL. If the ALTD instruction is present then the data in DE is exchanged with the data in the alternate register HL'.
- EX DE',HL exchanges the data in the alternate register DE' with the data in HL. If the ALTD instruction is present then the data in DE' is exchanged with the data in the alternate 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 BC, DE, and HL, with the data in their respective alternate registers BC', DE', and HL'.

IDET

Opcode	Instruction	Clocks	Operation
5B	IDET	2	Performs "LD E,E", but if (EDMR && SU[0]) then the System Violation interrupt flag is set.

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

The IDET instruction asserts a System Mode Violation interrupt if System/User mode is enabled (by writing to the Enable dual Mode Register, EDMR) and the processor is currently in user mode.

Note that IDET has the same opcode value as LD E,E, and actually executes that opcode as well as the behavior described above. If IDET is prefixed by ALTD, the opcode LD E',E is executed and the special System/User mode behavior does not occur.

This instruction is implemented in the Rabbit 3000A.

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 HL, or
- the sum of the data in IX and a displacement value *d*, or
- the sum of the data in 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 IX.
- INC IY increments the data in 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 *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 *ss* (any of BC, DE, HL, or SP).

IOE
IOI

Opcode	Instruction	Clocks	Operation
DB	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 internal 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 be avoided by putting a NOP instruction between an I/O instruction and any of the aforementioned instructions. Dynamic C versions 6.57 and later will automatically compensate for the bug. This bug is not present in the Rabbit 3000.

Examples

The following instruction loads the contents of A 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)
```

IPSET 0
IPSET 1
IPSET 2
IPSET 3

Opcode	Instruction	Clocks	Operation
ED 46	IPSET 0	4 (2,2)	IP = {IP[5:0], 00}
ED 56	IPSET 1	4 (2,2)	IP = {IP[5:0], 01}
ED 4E	IPSET 2	4 (2,2)	IP = {IP[5:0], 10}
ED 5E	IPSET 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. IPSET 0 forms the lowest priority; IPSET 3 forms the highest priority. These instructions are privileged.

- IPSET 0: The IPSET 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.
- IPSET 1: The IPSET 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.
- IPSET 2: The IPSET 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.
- IPSET 3: The IPSET 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 interrupts are suppressed (except the RST instruction).

IPRES

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. This instruction is privileged.

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 PC. Thus the address of the next instruction fetched is the data in HL.
- JP (IX): The data in IX is loaded into PC. Thus the address of the next instruction fetched is the data in IX.
- JP (IY): The data in IY is loaded into PC. Thus the address of the next instruction fetched is the data in IY.
- JP mn: The 16-bit constant *mn* is loaded into PC. 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 PC. If the condition is false then PC increments normally.

The condition *f* is one of the following: NZ, Z flag not set; Z, Z flag set; NC, C flag not set; C, C flag set; LZ, L/V flag is not set; LO, L/V flag is set; P, S flag not set; M, S 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 PC.

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

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

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

LCALL *x,mn*

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 XPC is pushed onto the stack. Next 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 SP 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*
——	**LD (HL),*r***	**6 (2,1,3)**	**(HL) = *r***
77	LD (HL),A	6 (2,1,3)	(HL) = A
70	LD (HL),B	6 (2,1,3)	(HL) = B
71	LD (HL),C	6 (2,1,3)	(HL) = C
72	LD (HL),D	6 (2,1,3)	(HL) = D
73	LD (HL),E	6 (2,1,3)	(HL) = E
74	LD (HL),H	6 (2,1,3)	(HL) = H
75	LD (HL),L	6 (2,1,3)	(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 BC with the data in A.
- `LD (DE),A:` Loads the memory location whose address is the data in DE with the data in A.
- `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 *r* (any of the registers A, B, C, D, E, H, or L).

LD (HL+*d*),HL

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

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

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 L into the memory location whose address is the sum of the data in IX and a displacement value *d*. Then, loads the data in H into the memory location whose address is the sum of the data in 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 IX and a displacement value *d*.
- LD (IX+d),r: Loads the data in *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 IX plus a displacement value *d*.

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

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*

——
FD 77 *d*
FD 70 *d*
FD 71 *d*
FD 72 *d*
FD 73 *d*
FD 74 *d*
FD 75 *d*
	LD (IY+*d*),*r*
	LD (IY+*d*),A
	LD (IY+*d*),B
	LD (IY+*d*),C
	LD (IY+*d*),D
	LD (IY+*d*),E
	LD (IY+*d*),H
	LD (IY+*d*),L
		10 (2,2,2,1,3)
		10 (2,2,2,1,3)
		10 (2,2,2,1,3)
		10 (2,2,2,1,3)
		10 (2,2,2,1,3)
		10 (2,2,2,1,3)
		10 (2,2,2,1,3)
		10 (2,2,2,1,3)
			(IY + *d*) = *r*
			(IY + *d*) = A
			(IY + *d*) = B
			(IY + *d*) = C
			(IY + *d*) = D
			(IY + *d*) = E
			(IY + *d*) = H
			(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 L into the memory location whose address is the sum of the data in IY and a displacement value *d*. Then, loads the data in H into the memory location whose address is the sum of the data in 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 IY and a displacement value *d*.
- LD (IY+d),r: Loads the data in *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 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 A.
- `LD (mn),HL:` Loads the memory location whose address is *mn* with the data in L, then loads the memory location whose address is 1 plus *mn* with the data in H.
- `LD (mn),IX:` Loads the memory location whose address is *mn* with the low order byte of the data in 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 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 *ss* (any of 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 *ss*.

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

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 L into the memory location whose address is the sum of the data in SP and the displacement *n*. Then loads the data in the 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 IX into the memory location whose address is the sum of the data in 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.
- LD (SP+n),IY: Loads the low order byte of the data in IY into the memory location whose address is the sum of the data in 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 A with the data whose address in memory is:

- the data in BC, or
- the data in 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 A 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 appropriate External Interrupt address to form the 16-bit ISR starting address.
- LD A,IIR: Loads A 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 appropriate Internal Peripheral address to form the 16-bit ISR starting address for that peripheral.

LD A,XPC

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 A with the data in XPC. This instruction is privileged.

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 HL,(*mn*)	13 (2,2,2,2,1,2,2)	L = (*mn*); H = (*mn* + 1)
ED 7B *n m*	LD SP,(*mn*)	13 (2,2,2,2,1,2,2)	SP$_{(low)}$=(*mn*); SP$_{(high)}$=(*mn*+1)

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

ALTD		
F	**R**	**SP**
	•	

I/O	
S	**D**
•	

Description

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

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

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 *dd'* (any of the registers BC', DE', or HL') with the data in BC or 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 HL,*mn*	6 (2,2,2)	HL = *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 *dd* (any of BC, DE, HL, 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 A. 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 appropriate External Interrupt address to form the 16-bit ISR starting address.
- LD IIR,A: Loads the Internal Interrupt Register, IIR, with the data in A. 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 appropriate Internal Peripheral address to form the 16-bit ISR starting address for that peripheral.

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

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

LD HL,(SP+*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 L with the data whose address is the data in SP plus a displacement *d*. Then loads H with the data whose address is the data in SP plus a displacement *d* plus 1.

LD HL,IX
LD HL,IY

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 HL with the data in IX.
- LD HL,IY: Loads HL with the data in 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 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*)

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

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

I/O	
S	**D**

Description

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

LD IY,(*mn*)

Opcode	Instruction	Clocks	Operation
FD 2A *n m*	LD IY,(*mn*)	13*	$IY_{(low)}$ = (*mn*); $IY_{(high)}$ = (*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 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*)

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

- the data in HL, or
- the sum of the data in IX and a displacement *d*, or
- the sum of the data in 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 *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
r,g	A	B	C	D	E	H	L	LD *r,g*	2	*r = g*
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 *r* (any of the registers A, B, C, D, E, H, or L) with the data in *g* (any of the registers A, B, C, D, E, H, or L).

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

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 SP with the data in (a) HL, (b) the data in IX, or (c) the data in IY. These are privileged instructions.

LD XPC,A

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 XPC with the data in A. This instruction is privileged.

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)	repeat: (DE) = (HL); BC = BC - 1; DE = DE - 1; HL = HL - 1 until { BC == 0 }
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)	repeat: (DE) = (HL); BC = BC - 1; DE = DE + 1; HL = HL + 1 until { BC == 0 }

Flags							
S	Z				L/V		C
-	-				•		-

ALTD		
F	R	SP

I/O	
S	D
	•

Description

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

If any of these block move instructions are prefixed by IOI or IOE, the destination will be in the specified I/O space. Add 1 clock for each iteration for the prefix if the prefix is IOI (internal I/O). If the prefix is IOE, add 2 clocks plus the number of I/O wait states enabled. The V flag is cleared when BC transitions from 1 to 0. If the V flag is not cleared another step is performed for the repeating versions of the instructions. Interrupts can occur between different repeats, but not within an iteration equivalent to LDD or LDI. Return from the interrupt is to the first byte of the instruction which is the I/O prefix byte if there is one.

LDDSR
LDISR

Opcode	Instruction	Clocks	Operation
ED 98	LDDSR	6+7i (2,2,1, (2,3,2)i,1)	(DE) = (HL); BC = BC - 1; HL = HL - 1; repeat while BC != 0
ED 90	LDISR	6+7i (2,2,1, (2,3,2)i,1)	(DE) = (HL); BC = BC; HL = HL + 1; repeat while BC != 0

Flags							
S	Z				L/V		C
-	-				•		-

ALTD		
F	R	SP

I/O	
S	D
	•

Description

- LDDSR: While the data in BC does not equal 0, the memory location whose address is in DE is loaded with the data at the address in HL. The data in BC and HL (but not DE) is then decremented. This instruction then repeats until BC equals zero. If this instruction is prefixed by IOI or IOE, the destination will be in the specified I/O space.
- LDISR: While the data in BC does not equal 0, the memory location whose address is in DE is loaded with the data at the address in HL. The data in BC is then decremented and HL incremented (the data in DE remains unchanged). This instruction then repeats until BC equals zero. If this instruction is prefixed by IOI or IOE, the destination will be in the specified I/O space.

Add 1 clock for each iteration for the prefix if the prefix is IOI (internal I/O). If the prefix is IOE, add 2 clocks plus the number of I/O wait states enabled. The V flag is cleared when BC transitions from 1 to 0. If the V flag is not cleared another step is performed for the repeating versions of the instructions. Interrupts can occur between different repeats, but not within an iteration. Return from the interrupt is to the first byte of the instruction which is the I/O prefix byte if there is one.

These instructions are implemented in the Rabbit 3000A.

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

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

Note that the LDP instructions wrap around on a 64K page boundary. Since the LDP instruction operates on two-byte values, the second byte will wrap around and be written at the start of the page if you try to read or write across a page boundary. Thus, if you fetch or store at address 0xn,0xFFFF, you will get the bytes located at 0xn, 0xFFFF and 0xn,0x0000 instead of 0xn,0xFFFF and 0x(n+1),0x0000 as you might expect. Therefore, do not use LDP at any physical address ending in 0xFFFF.

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

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 A (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 L, and then loads the following memory location with the data in 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 low order byte of IX, and then loads the following memory location with the high order byte of 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 low order byte of IY, and then loads the following memory location with the high order byte of IY.

Note that the LDP instructions wrap around on a 64K page boundary. Since the LDP instruction operates on two-byte values, the second byte will wrap around and be written at the start of the page if you try to read or write across a page boundary. Thus, if you fetch or store at address 0xn,0xFFFF, you will get the bytes located at 0xn, 0xFFFF and 0xn,0x0000 instead of 0xn,0xFFFF and 0x(n+1),0x0000 as you might expect. Therefore, do not use LDP at any physical address ending in 0xFFFF.

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

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

Note that the LDP instructions wrap around on a 64K page boundary. Since the LDP instruction operates on two-byte values, the second byte will wrap around and be written at the start of the page if you try to read or write across a page boundary. Thus, if you fetch or store at address 0xn,0xFFFF, you will get the bytes located at 0xn, 0xFFFF and 0xn,0x0000 instead of 0xn,0xFFFF and 0x(n+1),0x0000 as you might expect. Therefore, do not use LDP at any physical address ending in 0xFFFF.

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

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

Note that the LDP instructions wrap around on a 64K page boundary. Since the LDP instruction operates on two-byte values, the second byte will wrap around and be written at the start of the page if you try to read or write across a page boundary. Thus, if you fetch or store at address 0xn,0xFFFF, you will get the bytes located at 0xn, 0xFFFF and 0xn,0x0000 instead of 0xn,0xFFFF and 0x(n+1)0x0000 as you might expect. Therefore, do not use LDP at any physical address ending in 0xFFFF.

LJP *x,mn*

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 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

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 PC and the XPC off of the stack, in order to return from an LCALL operation.

First, the low order byte of PC is loaded with the data whose address is in SP. Next, the high order byte of PC is loaded with the data whose address is one plus the data in SP and XPC is loaded with the data whose address is two plus the data in SP. Finally the value in SP is incremented by 3.

LSDR
LSIR
LSDDR
LSIDR

Opcode	Instruction	Clocks	Operation
ED F8	LSDR	6+7i (2,2,1,(2,3,2)i,1)	(DE) = (HL); BC = BC-1; DE = DE-1; HL = HL-1; repeat while BC != 0
ED F0	LSIR	6+7i (2,2,1,(2,3,2)i,1)	(DE) = (HL); BC = BC-1; DE = DE+1; HL = HL+1; repeat while BC != 0
ED D8	LSDDR	6+7i (2,2,1,(2,3,2)i,1)	(DE) = (HL); BC = BC-1; DE = DE-1; repeat while BC != 0
ED D0	LSIDR	6+7i (2,2,1,(2,3,2)i,1)	(DE) = (HL); BC = BC-1; DE = DE+1; repeat while BC != 0

Flags							
S	Z				L/V		C
-	-				•		-

ALTD		
F	R	SP

I/O	
S	D
•	

Description

- LSDR: While the data in BC does not equal 0, the memory location whose address is in DE is loaded with the data at the address in HL. The data in BC, DE, and HL is then decremented. This instruction then repeats until BC equals zero. If this instruction is prefixed by IOI or IOE, the source will be in the specified I/O space.
- LSIR: While the data in BC does not equal 0, the memory location whose address is in DE is loaded with the data at the address in HL. The data in BC is then decremented, and the data in DE and HL is incremented. This instruction then repeats until BC equals zero. If this instruction is prefixed by IOI or IOE, the source will be in the specified I/O space.
- LSDDR: While the data in BC does not equal 0, the memory location whose address is in DE is loaded with the data at the address in HL. The data in BC and DE (but not HL) is then decremented. This instruction then repeats until BC equals zero. If this instruction is prefixed by IOI or IOE, the source will be in the specified I/O space.
- LSIDR: While the data in BC does not equal 0, the memory location whose address is in DE is loaded with the data ta the address in HL. The data in BC is then decremented and DE incremented (the data in HL remains unchanged). This instruction then repeats until BC equals zero. If this instruction is prefixed by IOI or IOE, the source will be in the specified I/O space.

Add 1 clock for each iteration for the prefix if the prefix is IOI (internal I/O). If the prefix is IOE, add 2 clocks plus the number of I/O wait states enabled. The V flag is cleared when BC transitions from 1 to 0. If the V flag is not cleared another step is performed for the repeating versions of the instructions. Interrupts can occur between different repeats, but not within an iteration. Return from the interrupt is to the first byte of the instruction which is the I/O prefix byte if there is one. These instructions are implemented for the Rabbit 3000A.

MUL

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 signed 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 HL (bits 31 through 16) and BC (bits 15 through 0) registers.

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 A from zero and stores the result in A.

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 A and the byte whose address is (a) in HL, (b) the sum of the data in IX and a displacement *d*, or (c) the sum of the data in 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 A.

Example

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

```
OR (HL)
```

would result in A containing 1110 1101.

OR HL,DE

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 HL and the data in 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
OR IY,DE

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 IX and the data in DE. The result is stored in IX
- OR IY,DE: Performs a logical OR operation between the data in IY and the data in 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 A and the 8-bit constant *n*.
- OR r: Performs a logical OR operation between the byte in A and the byte in *r* (any of the registers A, B, C, D, E, H, or L).

The relative bits of each byte are compared (i.e., bit 1 of both bytes are compared, 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 A.

POP IP
POP IX
POP IY

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. This is a privileged instruction.
- POP IX: Loads the low order byte of 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 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 SU

Opcode	Instruction	Clocks	Operation
ED 6E	POP SU	9 (2,2,2,3)	SU = (SP); SP = SP + 1

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

Loads the System/User Mode Register SU with the data at the memory location in SP, then increments the data in SP.

This instruction is privileged and is implemented in the Rabbit 3000A.

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 *zz* (any of AF, BC, DE, or HL) with the data at the memory address in 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

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 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 IY, and loads the memory location with the address two less than the data in SP with the low order byte of the data in IY. Then SP is decremented twice.

PUSH SU

Opcode	Instruction	Clocks	Operation
ED 66	PUSH SU	9 (2,2,2,3)	(SP - 1) = SU; SP = SP - 1

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

Loads the location in memory whose address is 1 less than the data held in SP with the data in the System/User Mode Register (SU) then decrements SP.

This instruction is privileged and is implemented in the Rabbit 3000A.

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 SP with the high order byte of the data in *zz* (any of 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.

RDMODE

Opcode	Instruction	Clocks	Operation
ED 7F	RDMODE	4 (2,2)	CF = SU[0]

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

The RDMODE instruction sets the C flag to the value of bit 0 of the System/User Mode Register (SU).

This instruction is implemented in the Rabbit 3000A.

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 HL, or
- the sum of the data in IX and a displacement *d*, or
- the sum of the data in IY 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
b,r	A	B	C	D	E	H	L	RES *b,r*	4 (2,2)	*r* = *r* & ~bit
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 held in *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 transfers execution from a subroutine to the program that called it. First it loads the low order byte of PC with the data at the memory address in 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 PC with the data at the memory address in 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** Z flag not set
- **Z** Z flag set
- **NC** C flag not set
- **C** C flag set
- **LZ/NV** L/V flag is not set
- **LO/V** L/V flag is set
- **P** S flag not set
- **M** S flag set.

RETI

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 SP. Then loads the low order byte of PC with the data whose address is 1 higher than the data in SP and loads the high order byte of PC with the data whose address is two higher than the data in SP. The data in SP is then incremented three times. This is a privileged instruction.

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 C flag the data whose address is:

- the data in HL, or
- the sum of the data in IX and a displacement *d*, or
- the sum of the data in 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 C flag moves to bit 0 and bit 7 moves to the C flag. See Figure 1 below.

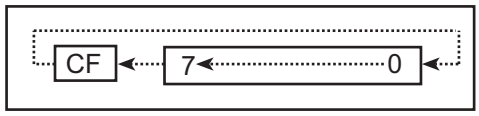


Figure 1: The bit logic of the RL instruction.

Example

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

```
RL (HL)
```

the byte in memory location 0x4545 will contain 1101 0101 and the C flag will be reset.

RL DE

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 C flag 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 C flag moves to bit 0 and bit 15 moves to the C flag. See figure below.

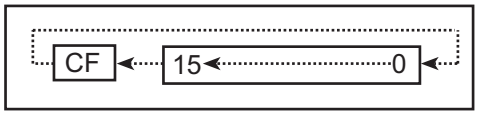


Figure 2: Bit logic of the RL instruction.

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 C flag the contents of *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 C flag moves to bit 0 and bit 7 moves to the C flag. See Figure 1 on page 115.

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 C flag the contents of A. Each bit in the register moves to the next highest-order bit position (bit 0 moves to bit 1, etc.) while the C flag moves to bit 0 and bit 7 moves to the C flag. See Figure 1 on page 115.

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 HL, or
- the sum of the data in IX and a displacement *d*, or
- the sum of the data in 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 C flag. See figure below.

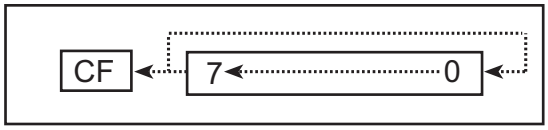


Figure 3: The bit logic of the RLC instruction.

Example

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

```
RLC (HL)
```

the byte in memory location 0x4545 will contain 1101 0100 and the C flag 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 *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 C flag. See Figure 3 on page 119.

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 A. 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 C flag. See Figure 3 on page 119.

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 C flag the data whose address is:

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

Bit 0 moves to the C flag, bits 1 through 7 move to the next lowest-order bit position, and the C flag moves to bit 7. See figure below.

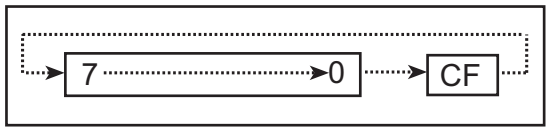


Figure 4: The bit logic for the RR instruction.

RR DE
RR HL

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 C flag the data in DE or HL. Bit 0 moves to the C flag, bits 1 through 15 move to the next lowest-order bit position, and the C flag moves to bit 15 (see figure below).

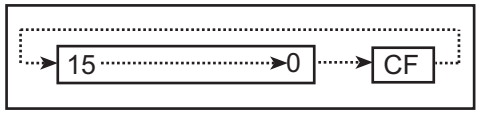


Figure 5: The bit logic for the RR instruction.

RR IX
RR IY

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 C flag the data in IX or IY. Bit 0 moves to the C flag, bits 1 through 15 move to the next lowest-order bit position, and the C flag moves to bit 15. See Figure 5 on page 123.

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 C flag the data in register *r* (any of the registers A, B, C, D, E, H, or L). Bit 0 moves to the C flag, bits 1 through 7 move to the next lowest-order bit position, and the C flag moves to bit 7. See Figure 4 on page 122.

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 C flag the data in A. Bit 0 moves to the C flag, bits 1 through 7 move to the next lowest-order bit position, and the C flag moves to bit 7. See Figure 4 on page 122.

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 HL, or
- the sum of the data in IX and a displacement *d*, or
- the sum of the data in 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 C flag. See figure below.

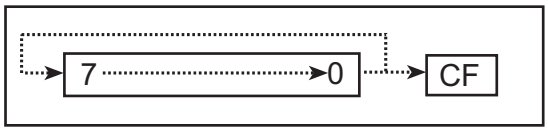


Figure 6: The bit logic of the RRC instruction.

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 *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 C flag. See Figure 6 on page 127.

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 A. 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 C flag. See Figure 6 on page 127.

RST v

Opcode	Instruction	Clocks	Operation
——	**RST *v***	**8 (2,2,2,2)**	**(SP - 1) = $PC_{(high)}$; (SP - 2) = $PC_{(low)}$; SP = SP - 2; PC = Restart Address**
D7	RST 10	8 (2,2,2,2)	{IIR, 0x20}
DF	RST 18	8 (2,2,2,2)	{IIR, 0x30}
E7	RST 20	8 (2,2,2,2)	{IIR, 0x40}
EF	RST 28	8 (2,2,2,2)	{IIR, 0x50}
FF	RST 38	8 (2,2,2,2)	{IIR, 0x70}

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

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

I/O	
S	**D**

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 SP. The value in SP is then decremented twice.

The PC is reset by loading it with the address to reset to v (any of the addresses 0020, 0030, 0040, 0050, or 0070).

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 C flag and the data whose address is:

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

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

These operations output an inverted carry:

- The C flag is set if A is less than the data being subtracted from it.
- The C flag is cleared if A is greater than the data being subtracted from it.
- The C flag is unchaged if A is equal to the data being subracted 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 C flag and the 8-bit constant *n* from the data in A.
- SBC A,r: Subtracts the C flag and the data in *r* (any of the registers A, B, C, D, E, H, or L) from the data in A.

The difference is stored in A.

These operations output an inverted carry:

- The C flag is set if A is less than the data being subtracted from it.
- The C flag is cleared if A is greater than the data being subtracted from it.
- The C flag is unchaged if A is equal to the data being subracted 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 C flag and the data in *ss* (any of BC, DE, HL, or SP) from the data in HL. The difference is stored in HL.

These operations output an inverted carry:

- The C flag is set if A is less than the data being subtracted from it.
- The C flag is cleared if A is greater than the data being subtracted from it.
- The C flag is unchaged if A is equal to the data being subracted 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 C flag.

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*,(IY+*d*)	13**	(IY + *d*) = (IY + *d*) | bit
FD CB *d* C6	SET bit 0,(IY+*d*)	13**	(IY + *d*) = (IY + *d*) | bit 0
FD CB *d* CE	SET bit 1,(IY+*d*)	13**	(IY + *d*) = (IY + *d*) | bit 1
FD CB *d* D6	SET bit 2,(IY+*d*)	13**	(IY + *d*) = (IY + *d*) | bit 2
FD CB *d* DE	SET bit 3,(IY+*d*)	13**	(IY + *d*) = (IY + *d*) | bit 3
FD CB *d* E6	SET bit 4,(IY+*d*)	13**	(IY + *d*) = (IY + *d*) | bit 4
FD CB *d* EE	SET bit 5,(IY+*d*)	13**	(IY + *d*) = (IY + *d*) | bit 5
FD CB *d* F6	SET bit 6,(IY+*d*)	13**	(IY + *d*) = (IY + *d*) | bit 6
FD CB *d* FE	SET 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

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 HL, or
- the sum of the data in IX and a displacement *d*, or
- the sum of the data in 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 *r* (any of the registers A, B, C, D, E, H, or L).

SETUSR

Opcode	Instruction	Clocks	Operation
ED 6F	SETUSR	4 (2,2)	SU={SU[5:0],0x01}

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

The System/User Mode Register, SU, is an 8 bit register that forms a stack of the current processor mode and the previous 3 modes. SETUSR shifts the contents of SU 2 bits to the left, then sets bit 1 to 0 and bit 0 to 1, signifying user mode.

This instruction is privileged and only implemented for the Rabbit 3000A.

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 HL, or
- the sum of the data in IX and a displacement *d*, or
- the sum of the data in 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 C flag. Bit 0 is reset. See figure below.

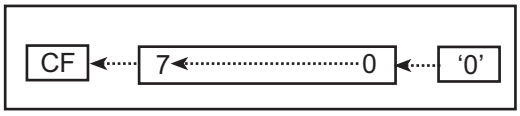


Figure 7: The bit logic of the SLA instruction.

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 C flag. Bit 0 is reset. See Figure 7 on page 138.

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 HL, or
- the sum of the data in IX and a displacement *d*, or
- the sum of the data in 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 C flag. See figure below.

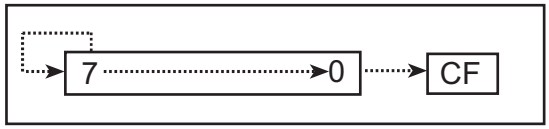


Figure 8: The bit logic of the SRA instruction.

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 *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 C flag. See Figure 8 on page 140.

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 HL, or
- the sum of the data in IX and a displacement *d*, or
- the sum of the data in 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 C flag. Bit 7 is reset. See figure below.

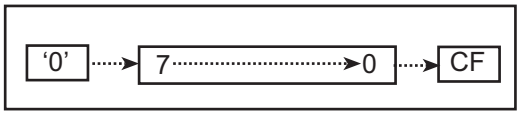


Figure 9: The bit logic of the SRL instruction.

SRL *r*

Opcode	Instruction	Clocks	Operation
——	**SRL *r***	**4 (2,2)**	***r* = {0,*r*[7,1]}; CF = *r*[0]**
CB 3F	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 *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 C flag. Bit 7 is reset. See Figure 9 on page 142.

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 A the data whose address is

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

The result is stored in A.

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 A the 8-bit constant *n*. The result is stored in A.

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

SURES

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

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

The SURES instruction rotates the contents of the System/User Mode Register SU 2 bits to the right, replacing the current processor mode with the previous mode.

This instruction is privileged and only implemented for the Rabbit 3000A.

SYSCALL

Opcode	Instruction	Clocks	Operation
ED 75	SYSCALL	10 (2,2,3,3)	SP = SP-2; PC = {R,v} where v = SYSCALL offset

Flags							
S	Z				L/V		C
-	-				-		-

ALTD		
F	R	SP

I/O	
S	D

Description

Pushes the current PC onto the stack and then resets the PC to the interrupt vector address represented by **IIR:0x60**, where **IIR** is the address of the interrupt table and **0x60** 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 SP. Then the low-order byte of the PC is loaded into the memory location with the address two less than the number in SP. The value in SP is then decremented twice.

This instruction is implemented in the Rabbit 3000A.

UMA
UMS

Opcode	Instruction	Clocks	Operation
ED C0	UMA	8+8i (2,2,2, (2,2,3,1)i,2)	[CY:DE':(HL)} = (IX) + [(IY)*DE+DE'+CY]; BC = BC - 1; IX = IX + 1; IY = IY + 1; HL = HL + 1; repeat while BC != 0
ED C8	UMS	8+8i (2,2,2, (2,2,3,1)i,2)	[CY:DE':(HL)} = (IX) - [(IY)*DE+DE'+CY]; BC = BC - 1; IX = IX + 1; IY = IY + 1; HL = HL + 1; repeat while BC != 0

Flags							
S	Z				L/V		C
-	-				-		•

ALTD		
F	R	SP

I/O	
S	D

Description

While the data in the BC does not equal 0, then:

- the data at the address in IY is multiplied by the data in DE;
- the data in alternate register DE' is added to that value;
- the C flag is added to that value; and
- this value is added to the data at the address in IX (for UMA) *or*
  this value is subtracted from the data at the address in IX (for UMS).

This results in a 24-bit value. The lowest eight bits of this value are stored memory at the address in HL, and the upper 16 bits are stored in the alternate register DE'. If The data in IX, IY, and HL are then incremented, and the data in BC is decremented. The instruction then repeats until BC equals zero. Interrupts can occur between different repeats, but not within an iteration.

These instructions are implemented in the Rabbit 3000A.

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 A and the data whose address is:

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

The corresponding bits of each byte are compared (i.e., bit 0 of both bytes are compared, bit 1 of both bytes are compared, etc.). 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 A.

Example

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

```
XOR (HL)
```

would result in the byte in A 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 A and the 8-bit constant *n*. The corresponding bits of each byte are compared (i.e., bit 0 of both bytes are compared, the bit 1 of both bytes are compared, etc.). 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 A.

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 A and *r* (any of the registers A, B, C, D, E, H, or L). The corresponding bits of each byte are compared (i.e., bit 0 of both bytes are compared, bit 1 of both bytes are compared, etc.). 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 A.

6. Opcode Map

Table 1: Main Page

LSB SB\	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	LD BC,mn	LD (BC),A	INC BC	INC B	DEC B	LD B,n	RLCA	EX AF,AF'	ADD HL,BC	LD A,(BC)	DEC BC	INC C	DEC C	LD C,n	RRCA
1	DJNZ e	LD DE,mn	LD (DE),A	INC DE	INC D	DEC D	LD D,n	RLA	JR e	ADD HL,DE	LD A,(DE)	DEC DE	INC E	DEC E	LD E,n	RRA
2	JR NZ,e	LD HL,mn	LD (mn),HL	INC HL	INC H	DEC H	LD H,n	ADD SP,d	JR Z,e	ADD HL,HL	LD HL,(mn)	DEC HL	INC L	DEC L	LD L,n	CPL
3	JR NC,e	LD SP,mn	LD (mn),A	INC SP	INC (HL)	DEC (HL)	LD (HL),n	SCF	JR C,e	ADD HL,SP	LD A,(mn)	DEC SP	INC A	DEC A	LD A,n	CCF
4	LD B,B	LD B,C	LD B,D	LD B,E	LD B,H	LD B,L	LD B,(HL)	LD B,A	LD C,B	LD C,C	LD C,D	LD C,E	LD C,H	LD C,L	LD C,(HL)	LD C,A
5	LD D,B	LD D,C	LD D,D	LD D,E	LD D,H	LD D,L	LD D,(HL)	LD D,A	LD E,B	LD E,C	LD E,D	LD E,E (IDET)	LD E,H	LD E,L	LD E,(HL)	LD E,A
6	LD H,B	LD H,C	LD H,D	LD H,E	LD H,H	LD H,L	LD H,(HL)	LD H,A	LD L,B	LD L,C	LD L,D	LD L,E	LD L,H	LD L,L	LD L,(HL)	LD L,A
7	LD (HL),B	LD (HL),C	LD (HL),D	LD (HL),E	LD (HL),H	LD (HL),L	ALTD	LD (HL),A	LD A,B	LD A,C	LD A,D	LD A,E	LD A,H	LD A,L	LD A,(HL)	LD A,A
8	ADD A,B	ADD A,C	ADD A,D	ADD A,E	ADD A,H	ADD A,L	ADD A,(HL)	ADD A,A	ADC A,B	ADC A,C	ADC A,D	ADC A,E	ADC A,H	ADC A,L	ADC A,(HL)	ADC A,A
9	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB (HL)	SUB A	SBC A,B	SBC A,C	SBC A,D	SBC A,E	SBC A,H	SBC A,L	SBC A,(HL)	SBC A,A
A	AND B	AND C	AND D	AND E	AND H	AND L	AND (HL)	AND A	XOR B	XOR C	XOR D	XOR E	XOR H	XOR L	XOR (HL)	XOR A
B	OR B	OR C	OR D	OR E	OR H	OR L	OR (HL)	OR A	CP B	CP C	CP D	CP E	CP H	CP L	CP (HL)	CP A
C	RET NZ	POP BC	JP NZ,mn	JP mn	LD HL, (SP+n)	PUSH BC	ADD A,n	LJP x,mn	RET Z	RET	JP Z,mn	esc	BOOL HL	CALL nn	ADC A,n	LCALL x,mn
D	RET NC	POP DE	JP NC,mn	IOI	LD (SP+n), HL	PUSH DE	SUB n	RST 10	RET C	EXX	JP C,mn	IOE	AND HL,DE	esc	SBC A,n	RST 18
E	RET PO	POP HL	JP PO,mn	EX DE',HL	LD HL, (IX+d)	PUSH HL	AND n	RST 20	RET PE	JP (HL)	JP PE,mn	EX DE,HL	OR HL,DE	esc	XOR n	RST 28
F	RET P	POP AF	JP P,mn	RL DE	LD (IX+d), HL	PUSH AF	OR n	MUL	RET M	LD SP,HL	JP M,mn	RR DE	RR HL	esc	CP n	RST 38

Table 2: ED Page

\LSB MSB\	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2																
3																
4		LD BC',DE	SBC HL,BC	LD (mn),BC	NEG	LRET	IPSET 0	LD I,A		LD BC',BC	ADC HL,BC	LD BC,(mn)		RETI	IPSET 2	LD R,A
5		LD DE',DE	SBC HL,DE	LD (mn),DE	EX (SP),HL		IPSET 1	LD A,I		LD DE',BC	ADC HL,DE	LD DE,(mn)		IPRES	IPSET 3	LD A,R
6		LD HL',DE	SBC HL,HL	LD (mn),HL	LDP (HL),HL	LDP (mn),HL	PUSH SU	LD XPC,A		LD HL',BC	ADC HL,HL	LD HL,(mn)	LDP HL,(HL)	LDP HL,(mn)	POP SU	SETUSR
7			SBC HL,SP	LD (mn),SP		SYSCALL	PUSH IP	LD A,XPC			ADC HL,SP	LD SP,(mn)		SURES	POP IP	RDMODE
8																
9	LDISR								LDDSR							
A	LDI								LDD							
B	LDIR								LDDR							
C	UMA								UMS							
D	LSIDR								LSDDR							
E																
F	LSIR								LSDR							

Table 3: DD Page

\LSB MSB\	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0										ADD IX,BC						
1										ADD IX,DE						
2		LD IX,mn	LD (mn),IX	INC IX						ADD IX,IX	LD IX,(mn)	DEC IX				
3					INC (IX+d)	DEC (IX+d)	LD (IX+d),n			ADD IX,SP						
4							LD B,(IX+d)								LD C,(IX+d)	
5							LD D,(IX+d)								LD E,(IX+d)	
6					LDP (IX),HL	LDP (mn),IX	LD H,(IX+d)						LDP HL,(IX)	LDP IX,(mn)	LD L,(IX+d)	
7	LD (IX+d), B	LD (IX+d), C	LD (IX+d), D	LD (IX+d), E	LD (IX+d), H	LD (IX+d), L		LD (IX+d), A					LD HL,IX	LD IX,HL	LD A,(IX+d)	
8							ADD A,(IX+d)								ADC A,(IX+d)	
9							SUB (IX+d)								SBC A,(IX+d)	
A							AND (IX+d)								XOR (IX+d)	
B							OR (IX+d)								CP (IX+d)	
C					LD IX, (SP+n)							esc	BOOL IX			
D					LD (SP+n), IX								AND IX,DE			
E		POP IX		EX (SP),IX	LD HL, (HL+d)	PUSH IX				JP (IX)			OR IX,DE			
F					LD (HL+d), HL					LD SP,IX			RR IX			

Table 4: FD Page

\LSB MSB\	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0										ADD IY,BC						
1										ADD IY,DE						
2		LD IY,mn	LD (mn),IY	INC IY						ADD IY,IY	LD IY,(mn)	DEC IY				
3					INC (IY+d)	DEC (IY+d)	LD (IY+d),n			ADD IX,SP						
4							LD B,(IY+d)								LD C,(IY+d)	
5							LD D,(IY+d)								LD E,(IY+d)	
6					LDP (IY),HL	LDP (mn),IY	LD H,(IY+d)						LDP HL,(IY)	LDP IY,(mn)	LD L,(IY+d)	
7	LD (IY+d), B	LD (IY+d), C	LD (IY+d), D	LD (IY+d), E	LD (IY+d), H	LD (IY+d), L		LD (IY+d), A					LD HL,IY	LD IY,HL	LD A,(IY+d)	
8							ADD A,(IY+d)								ADC A,(IY+d)	
9							SUB (IY+d)								SBC A,(IY+d)	
A							AND (IY+d)								XOR (IY+d)	
B							OR (IY+d)								CP (IY+d)	
C					LD IY, (SP+n)							esc	BOOL IY			
D					LD (SP+n), IY								AND IY,DE			
E		POP IY		EX (SP),IY	LD HL, (IY+d)	PUSH IY				JP (IY)			OR IY,DE			
F					LD (IY+d), HL					LD SP,IY			RR IY			

Table 5: CB Page

SB SB\	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	RLC B	RLC C	RLC D	RLC E	RLC H	RLC L	RLC (HL)	RLC A	RRC B	RRC C	RRC D	RRC E	RRC H	RRC L	RRC (HL)	RRC A
1	RL B	RL C	RL D	RL E	RL H	RL L	RL (HL)	RL A	RR B	RR C	RR D	RR E	RR H	RR L	RR (HL)	RR A
2	SLA B	SLA C	SLA D	SLA E	SLA H	SLA L	SLA (HL)	SLA A	SRA B	SRA C	SRA D	SRA E	SRA H	SRA L	SRA (HL)	SRA A
3									SRL B	SRL C	SRL D	SRL E	SRL H	SRL L	SRL (HL)	SRL A
4	BIT 0,B	BIT 0,C	BIT 0,D	BIT 0,E	BIT 0,H	BIT 0,L	BIT 0,(HL)	BIT 0,A	BIT 1,B	BIT 1,C	BIT 1,D	BIT 1,E	BIT 1,H	BIT 1,L	BIT 1,(HL)	BIT 1,A
5	BIT 2,B	BIT 2,C	BIT 2,D	BIT 2,E	BIT 2,H	BIT 2,L	BIT 2,(HL)	BIT 2,A	BIT 3,B	BIT 3,C	BIT 3,D	BIT 3,E	BIT 3,H	BIT 3,L	BIT 3,(HL)	BIT 3,A
6	BIT 4,B	BIT 4,C	BIT 4,D	BIT 4,E	BIT 4,H	BIT 4,L	BIT 4,(HL)	BIT 4,A	BIT 5,B	BIT 5,C	BIT 5,D	BIT 5,E	BIT 5,H	BIT 5,L	BIT 5,(HL)	BIT 5,A
7	BIT 6,B	BIT 6,C	BIT 6,D	BIT 6,E	BIT 6,H	BIT 6,L	BIT 6,(HL)	BIT 6,A	BIT 7,B	BIT 7,C	BIT 7,D	BIT 7,E	BIT 7,H	BIT 7,L	BIT 7,(HL)	BIT 7,A
8	RES 0,B	RES 0,C	RES 0,D	RES 0,E	RES 0,H	RES 0,L	RES 0,(HL)	RES 0,A	RES 1,B	RES 1,C	RES 1,D	RES 1,E	RES 1,H	RES 1,L	RES 1,(HL)	RES 1,A
9	RES 2,B	RES 2,C	RES 2,D	RES 2,E	RES 2,H	RES 2,L	RES 2,(HL)	RES 2,A	RES 3,B	RES 3,C	RES 3,D	RES 3,E	RES 3,H	RES 3,L	RES 3,(HL)	RES 3,A
A	RES 4,B	RES 4,C	RES 4,D	RES 4,E	RES 4,H	RES 4,L	RES 4,(HL)	RES 4,A	RES 5,B	RES 5,C	RES 5,D	RES 5,E	RES 5,H	RES 5,L	RES 5,(HL)	RES 5,A
B	RES 6,B	RES 6,C	RES 6,D	RES 6,E	RES 6,H	RES 6,L	RES 6,(HL)	RES 6,A	RES 7,B	RES 7,C	RES 7,D	RES 7,E	RES 7,H	RES 7,L	RES 7,(HL)	RES 7,A
C	SET 0,B	SET 0,C	SET 0,D	SET 0,E	SET 0,H	SET 0,L	SET 0,(HL)	SET 0,A	SET 1,B	SET 1,C	SET 1,D	SET 1,E	SET 1,H	SET 1,L	SET 1,(HL)	SET 1,A
D	SET 2,B	SET 2,C	SET 2,D	SET 2,E	SET 2,H	SET 2,L	SET 2,(HL)	SET 2,A	SET 3,B	SET 3,C	SET 3,D	SET 3,E	SET 3,H	SET 3,L	SET 3,(HL)	SET 3,A
E	SET 4,B	SET 4,C	SET 4,D	SET 4,E	SET 4,H	SET 4,L	SET 4,(HL)	SET 4,A	SET 5,B	SET 5,C	SET 5,D	SET 5,E	SET 5,H	SET 5,L	SET 5,(HL)	SET 5,A
F	SET 6,B	SET 6,C	SET 6,D	SET 6,E	SET 6,H	SET 6,L	SET 6,(HL)	SET 6,A	SET 7,B	SET 7,C	SET 7,D	SET 7,E	SET 7,H	SET 7,L	SET 7,(HL)	SET 7,A

Table 6: DD-CB Page

B B\	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
							RLC (IX+d)								RRC (IX+d)	
							RL (IX+d)								RR (IX+d)	
							SLA (IX+d)								SRA (IX+d)	
															SRL (IX+d)	
							BIT 0,(IX+d)								BIT 1,(IX+d)	
							BIT 2,(IX+d)								BIT 3,(IX+d)	
							BIT 4,(IX+d)								BIT 5,(IX+d)	
							BIT 6,(IX+d)								BIT 7,(IX+d)	
							RES 0,(IX+d)								RES 1,(IX+d)	
							RES 2,(IX+d)								RES 3,(IX+d)	
							RES 4,(IX+d)								RES 5,(IX+d)	
							RES 6,(IX+d)								RES 7,(IX+d)	
							SET 0,(IX+d)								SET 1,(IX+d)	
							SET 2,(IX+d)								SET 3,(IX+d)	
							SET 4,(IX+d)								SET 5,(IX+d)	
							SET 6,(IX+d)								SET 7,(IX+d)	

Table 7: FD-CB Page

B B\	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
							RLC (IY+d)								RRC (IY+d)	
							RL (IY+d)								RR (IY+d)	
							SLA (IY+d)								SRA (IY+d)	
															SRL (IY+d)	
							BIT 0,(IY+d)								BIT 1,(IY+d)	
							BIT 2,(IY+d)								BIT 3,(IY+d)	
							BIT 4,(IY+d)								BIT 5,(IY+d)	
							BIT 6,(IY+d)								BIT 7,(IY+d)	
							RES 0,(IY+d)								RES 1,(IY+d)	
							RES 2,(IY+d)								RES 3,(IY+d)	
							RES 4,(IY+d)								RES 5,(IY+d)	
							RES 6,(IY+d)								RES 7,(IY+d)	
							SET 0,(IY+d)								SET 1,(IY+d)	
							SET 2,(IY+d)								SET 3,(IY+d)	
							SET 4,(IY+d)								SET 5,(IY+d)	
							SET 6,(IY+d)								SET 7,(IY+d)	

7. Quick Reference Table

Key

- **`Instruction`**: The mnemonic syntax of the instruction.
- **`Opcode`**: The binary bytes that represent the instruction.
- **`Clock cycles`**: The number of clock cycles that the instruction takes to complete. The numbers in parenthesis are a breakdown of the total clocks. For more information, please see Table 1: "Typical Clocks Breakdown" on page 7.
- **`A`**: How the ALTD prefix affects the instruction. For more information, please see Table 2: "ALTD (“A” Column) Symbol Key" on page 8.
- **`I`**: How the IOI or IOE prefixes affect the instruction. For more information, please see Table 3: "IOI and IOE (“I” Column) Symbol Key" on page 8. 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. For more information, please see Table 4: "Flag Register Key" on page 8.
- **`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 2000/3000. 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

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
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
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	00ss1011				2	r		-	-	-	-	ss = ss - 1	
DJNZ e	00010000	--(e-2)-			5 (2,2,1)	r		-	-	-	-	B = B-1; if {B != 0} PC = PC + e	
EX (SP),HL	11101101	01010100			15 (2,2,1,2,2,3,3)	r		-	-	-	-	H <-> (SP+1); L <-> (SP)	M
EX (SP),IX	11011101	11100011			15 (2,2,1,2,2,3,3)			-	-	-	-	IXH <-> (SP+1); IXL <-> (SP)	
EX (SP),IY	11111101	11100011			15 (2,2,1,2,2,3,3)			-	-	-	-	IYH <-> (SP+1); IYL <-> (SP)	
EX AF,AF'	00001000				2			-	-	-	-	AF <-> AF'	
EX DE,HL	11101011				2	s		-	-	-	-	if (!ALTD) then DE <-> HL else DE <-> HL'	N
EX DE',HL	11100011				2	s		-	-	-	-	if (!ALTD) then DE' <-> HL else DE' <-> HL'	
EXX	11011001				2			-	-	-	-	BC <-> BC'; DE <-> DE'; HL <-> HL'	
IDET	01011011				2			-	-	-	-	E = E, but if (EDMR && SU[0]) then System Violation interrupt flag is set	
INC (HL)	00110100				8 (2,1,2,3)	f	b	*	*	V	-	(HL) = (HL) + 1	
INC (IX+d)	11011101	00110100	----d---		12 (2,2,2,1,2,3)	f	b	*	*	V	-	(IX+d) = (IX+d) + 1	
INC (IY+d)	11111101	00110100	----d---		12 (2,2,2,1,2,3)	f	b	*	*	V	-	(IY+d) = (IY+d) + 1	
INC IX	11011101	00100011			4 (2,2)			-	-	-	-	IX = IX + 1	
INC IY	11111101	00100011			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
IPSET 0	11101101	01000110			4 (2,2)			-	-	-	-	IP = {IP[5:0], 00}	NP
IPSET 1	11101101	01010110			4 (2,2)			-	-	-	-	IP = {IP[5:0], 01}	NP

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
IPSET 2	11101101	01001110			4 (2,2)			-	-	-	-	IP = {IP[5:0], 10}	NP
IPSET 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) = PCL; (SP-2) = PCH; (SP-3) = XPC; 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	----n---	----m---		11 (2,2,2,1,2,2)	r	s	-	-	-	-	L = (mn); H = (mn+1)	
LD HL,(HL+d)	11011101	11100100	----d---		11 (2,2,2,1,2,2)	r	s	-	-	-	-	L = (HL+d); H = (HL+d+1)	N
LD HL,(IX+d)	11100100	----d---			9 (2,2,1,2,2)	r	s	-	-	-	-	L = (IX+d); H = (IX+d+1)	N
LD HL,(IY+d)	11111101	11100100	----d---		11 (2,2,2,1,2,2)	r	s	-	-	-	-	L = (IY+d); H = (IY+d+1)	N

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
LD HL,(SP+n)	11000100	----n---			9 (2,2,1,2,2)	r		-	-	-	-	L = (SP+n); H = (SP+n+1)	N
LD HL,IX	11011101	01111100			4 (2,2)	r		-	-	-	-	HL = IX	N
LD HL,IY	11111101	01111100			4 (2,2)	r		-	-	-	-	HL = IY	N
LD IX,(mn)	11011101	00101010	----n---	----m---	13 (2,2,2,2,1,2,2)		s	-	-	-	-	IXL = (mn); IXH = (mn+1)	
LD IX,(SP+n)	11011101	11000100	----n---		11 (2,2,2,1,2,2)			-	-	-	-	IXL = (SP+n); IXH = (SP+n+1)	N
LD IX,HL	11011101	01111101			4 (2,2)			-	-	-	-	IX = HL	N
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--g				2	r		-	-	-	-	r = g	
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	-	-	*	-	repeat: (DE) = (HL); BC = BC-1; DE = DE-1; HL = HL-1 until {BC==0}	
LDDSR	11101101	10011000			6+7i (2,2,1,(2,3,2)i,1)		d	-	-	*	-	(DE) = (HL); BC = BC-1; HL = HL-1; repeat while BC != 0	
LDI	11101101	10100000			10 (2,2,1,2,3)		d	-	-	*	-	(DE) = (HL); BC = BC-1; DE = DE+1; HL = HL+1	
LDIR	11101101	10110000			6+7i (2,2,1,(2,3,2)i,1)		d	-	-	*	-	repeat: (DE) = (HL); BC = BC-1; DE = DE+1; HL = HL+1 until {BC == 0}	
LDISR	11101101	10010000			6+7i (2,2,1,(2,3,2)i,1)		d	-	-	*	-	(DE) = (HL); BC = BC-1; HL = HL+1; repeat while BC != 0	
LDP (HL),HL	11101101	01100100			12 (2,2,2,3,3)			-	-	-	-	(HL) = L; (HL+1) = H. (Addr[19:16] = A[3:0])	N
LDP (IX),HL	11011101	01100100			12 (2,2,2,3,3)			-	-	-	-	(IX) = L; (IX+1) = H. (Addr[19:16] = A[3:0])	N
LDP (IY),HL	11111101	01100100			12 (2,2,2,3,3)			-	-	-	-	(IY) = L; (IY+1) = H. (Addr[19:16] = A[3:0])	N
LDP (mn),HL	11101101	01100101	----n---	----m---	15 (2,2,2,2,1,3,3)			-	-	-	-	(mn) = L; (mn+1) = H. (Addr[19:16] = A[3:0])	N
LDP (mn),IX	11011101	01100101	----n---	----m---	15 (2,2,2,2,1,3,3)			-	-	-	-	(mn) = IXL; (mn+1) = IXH. (Addr[19:16] = A[3:0])	N
LDP (mn),IY	11111101	01100101	----n---	----m---	15 (2,2,2,2,1,3,3)			-	-	-	-	(mn) = IYL; (mn+1) = IYH. (Addr[19:16] = A[3:0])	N
LDP HL,(HL)	11101101	01101100			10 (2,2,2,2,2)			-	-	-	-	L = (HL); H = (HL+1). (Addr[19:16] = A[3:0])	N
LDP HL,(IX)	11011101	01101100			10 (2,2,2,2,2)			-	-	-	-	L = (IX); H = (IX+1). (Addr[19:16] = A[3:0])	N
LDP HL,(IY)	11111101	01101100			10 (2,2,2,2,2)			-	-	-	-	L = (IY); H = (IY+1). (Addr[19:16] = A[3:0])	N
LDP HL,(mn)	11101101	01101101	----n---	----m---	13 (2,2,2,2,1,2,2)			-	-	-	-	L = (mn); H = (mn+1). (Addr[19:16] = A[3:0])	N
LDP IX,(mn)	11011101	01101101	----n---	----m---	13 (2,2,2,2,1,2,2)			-	-	-	-	IXL = (mn); IXH = (mn+1). (Addr[19:16] = A[3:0])	N

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
LDP IY,(mn)	11111101	01101101	----n---	----m---	13 (2,2,2,2,1,2,2)			-	-	-	-	IYL = (mn); IYH = (mn+1). (Addr[19:16] = A[3:0])	N
LJP x,mn	11000111	----n---	----m---	---x----	10 (2,2,2,2,2)			-	-	-	-	XPC = x; PC = mn	N
LRET	11101101	01000101			13 (2,2,1,2,2,2,2)			-	-	-	-	PCL = (SP); PCH = (SP+1); XPC = (SP+2); SP = SP+3	N
LSDR	11101101	11111000			6+7i (2,2,1,(2,3,2)i,1)		s	-	-	*	-	(DE) = (HL); BC = BC-1; DE = DE-1; HL = HL-1; repeat while BC != 0	
LSDDR	11101101	11011000			6+7i (2,2,1,(2,3,2)i,1)		s	-	-	*	-	(DE) = (HL); BC = BC-1; DE = DE-1; repeat while BC != 0	
LSIDR	11101101	11010000			6+7i (2,2,1,(2,3,2)i,1)		s	-	-	*	-	(DE) = (HL); BC = BC-1; DE = DE+1; repeat while BC != 0	
LSIR	11101101	11110000			6+7i (2,2,1,(2,3,2)i,1)		s	-	-	*	-	(DE) = (HL); BC = BC-1; DE = DE+1; HL = HL+1; repeat while BC != 0	
MUL	11110111				12 (2,10)			-	-	-	-	HL:BC = BC * DE	N
NEG	11101101	01000100			4 (2,2)	fr		*	*	V	*	A = 0 - A	
NOP	00000000				2			-	-	-	-	No operation	
OR (HL)	10110110				5 (2,1,2)	fr	s	*	*	L	0	A = A | (HL)	
OR (IX+d)	11011101	10110110	----d---		9 (2,2,2,1,2)	fr	s	*	*	L	0	A = A | (IX+d)	
OR (IY+d)	11111101	10110110	----d---		9 (2,2,2,1,2)	fr	s	*	*	L	0	A = A | (IY+d)	
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 SU	11101101	01101110			9 (2,2,2,3)			-	-	-	-	SU = (SP); SP = SP+1	P
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 SU	11101101	01100110			9 (2,2,2,3)			-	-	-	-	(SP-1) = SU; SP = SP-1	P
PUSH zz	11zz0101				10 (2,2,3,3)			-	-	-	-	(SP-1) = zzh; (SP-2) = zzl; SP = SP-2	
RDMODE	11101101	01111111			4 (2,2)			-	-	-	•	CF = SU[0]	P
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]	

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
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]	
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 A,(HL)	11011101	10011110	----d---		9 (2,2,2,1,2)	fr	s	*	*	V	*	A = A - (IX+d) - CY	
SBC (IX+d)	11111101	10011110	----d---		9 (2,2,2,1,2)	fr	s	*	*	V	*	A = A - (IY+d) - CY	
SBC (IY+d)	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	
SETUSR	11101101	01101111			4 (2,2)			-	-	-	-	SU = {SU[5:0], 0x01}	P
SLA (HL)	11001011	00100110			10 (2,2,1,2,3)	f	b	*	*	L	*	(HL) = {(HL)[6,0],0}; CY = (HL)[7]	
SLA (IX+d)	11011101	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)	11111101	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	00101110			10 (2,2,1,2,3)	f	b	*	*	L	*	(HL) = {(HL)[7],(HL)[7,1]}; CY = (HL)[0]	
SRA (IX+d)	11011101	11001011	----d---	00101110	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)	11111101	11001011	----d---	00101110	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	00111110			10 (2,2,1,2,3)	f	b	*	*	L	*	(HL) = {0,(HL)[7,1]}; CY = (HL)[0]	
SRL (IX+d)	11011101	11001011	----d---	00111110	13 (2,2,2,2,2,3)	f	b	*	*	L	*	(IX+d) = {0,(IX+d)[7,1]}; CY = (IX+d)[0]	
SRL (IY+d)	11111101	11001011	----d---	00111110	13 (2,2,2,2,2,3)	f	b	*	*	L	*	(IY+d) = {0,(IY+d)[7,1]}; CY = (IY+d)[0]	
SRL r	11001011	00111-r-			4 (2,2)	fr		*	*	L	*	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
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	
SURES	11101101	01111101			4 (2,2)			-	-	-	-	SU = {SU[1:0],SU[7:2]}	P
SYSCALL	11101101	01110101			10 (2,2,3,3)			-	-	-	-	SP = SP-2; PC = {R,v} where v = SYSCALL offset	
UMA	11101101	11000000			8+8i (2,2,2,(2,2,3,1)i,2)			-	-	-	*	{CY:DE':(HL)} = (IX) + [(IY)*DE+DE'+CY]; BC = BC-1; IX = IX+1; IY = IY+1; HL = HL+1; repeat while BC != 0	
UMS	11101101	11001000			8+8i (2,2,2,(2,2,3,1)i,2)			-	-	-	*	{CY:DE':(HL)} = (IX) - [(IY)*DE+DE'+CY]; BC = BC-1; IX = IX+1; IY = IY+1; HL = HL+1; repeat while BC != 0	
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]	

Rabbit 2000/3000 Microprocessor Instruction Reference Manual

Part Number 019–0098 F • 040114 • 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