PRODUCT MANUAL

Rabbit Family of Microprocessors

Instruction Reference Manual

019–0098 • 090409-L

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

Rabbit Family of Microprocessors Instruction Set Reference Manual

Part Number 019-0098 • 090409–L • Printed in U.S.A.

Trademarks

Table of Contents

Rabbit Instructions Listed Alphabetically

All Rabbit processor instructions are listed below. Note that some instructions have two entries. The first one is for the Rabbit 2000/3000 version of the instruction. The second entry is for Rabbit 4000 and newer processors.

Rabbit Instructions Listed by Group

All Rabbit processor instructions are listed below by group. Note that some instructions have two entries, e.g., ADC A,r. The first one is for the Rabbit 2000/3000 version of the instruction. The second entry is for the Rabbit 4000 and newer processors.

Chapter 1. Definitions and Conventions

This chapter describes the formatting of information that explains the Rabbit assembly instructions. The symbols and condition codes used in the instruction mnemonics are listed and described. At the end of the chapter is a short list of definitions.

1.1 Instruction Table Key

For the most part, you will find three tables that explain an instruction. The Instruction table is defined by the following columns:

- **`Opcode`**: A hexadecimal representation of the value that the mnemonic instruction represents.
- **`Instruction`**: The mnemonic syntax of the instruction.
- **`Operation`**: A symbolic representation of the operation performed.

1.2 Clocks Table Key

The Clocks table states the number of clock cycles it takes to complete this instruction. The number of clocks instructions take follows a general pattern. 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.

The clocking of every instruction is affected by the type of memory access enabled for instruction fetches. The Rabbit 2000 and 3000 only allow 8-bit memory accesses for fetching instructions, while the Rabbit 4000 and 5000 allow either 8-bit or 16-bit memory accesses. Since the Rabbit processors have instructions of various lengths, part of the instruction may be unaligned with the 16-bit memory alignment, so instruction clocks are listed for instructions when they either start on a 16-bit boundary (even address, or aligned) or not (odd address, or unaligned).

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

1.3 Flags, ALTD, and IOI/IOE Table Keys

The second table for an instruction identifies how executing that instruction affects the flags register and also how the instruction is affected by the instruction prefixes ALTD, IOI and IOE.

Table 2: Flag Register Key

S	Z	L/V	C	Description
•				Sign flag affected; set if result is negative, cleared if result is positive
-				Sign flag not affected
	•			Zero flag affected; set if result is zero, cleared if result is not zero.
	-			Zero flag not affected
		L		Logical/Overflow flag contains logical check result; set if result is one, cleared if result is zero.
		V		Logical/Overflow flag set on arithmetic overflow result, cleared if there was no arithmetic overflow
		0		Logical/Overflow flag is cleared
		•		Logical/Overflow flag is affected
			•	Carry flag is affected
			-	Carry flag is not affected
			0	Carry flag is cleared
			1	Carry flag is set

Table 3: 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 4: IOI and IOE (“I” Column) Symbol Key

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

1.4 Memory Modes

There are two memory modes available in the Rabbit 2000 and Rabbit 3000: logical and physical. The Rabbit 4000 and newer Rabbit processors have three memory modes: logical, physical and pointer indirect.

1.5 Instruction Symbols Key

This table describes the symbols used in the instruction descriptions.

Table 5: Symbols Used in Instruction Descriptions

Symbol	Symbol Meaning
`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)
`bb`	Determines number of times (1, 2 or 4) to repeat certain rotate and shift instructions.
`cc`	Condition code select (00 = NZ, 01 = Z, 10 = NC, 11 = C)
`cx`	Condition code select (00 = GT, 01 = GTU, 10 = LT, 11 = V
`d`	8-bit signed integer, in the range [-128, 127]. Expressed in two's complement.
`dd`	16-bit register select-destination (00 = BC, 01 = DE, 10 = HL, 11 = SP)
`dd'`	16-bit register select-alternate(00 = BC', 01 = DE', 10 = HL')
`e`[a]	8-bit signed displacement in the range [-128, 127] added to PC
`ee`[b]	16-bit signed displacement in the range [-32768, 32767] added to PC
`f`	Condition code select (000 = NZ, 001 = Z, 010 = NC, 011 = C, 100 = LZ/NV, 101 = LO/V, 110 = P, 111 = M)
`m`	Most significant bits (MSB) of a 16-bit constant
`mn`	16-bit constant
`lmn`	24-bit constant
`lxpc`	12-bit XPC
`n`	8-bit constant or the least significant bits (LSB) of a 16-bit constant
`ps, pd`	32-bit register select: 1000 = PW, 1001 = PX, 1010 = PY, 1011 = PZ
`pp`	32-bit register select: 00 = PW, 01 = PX 10 = PY, 11 = PZ
`r, g`	8-bit register select: 000 = B, 001 = C, 010 = D, 011 = E, 100 = H, 101 = L, 111 = A
`rr`	16-bit register select: 00 = BC, 01 = DE, 10 = IX, 11 = IY
`ss`	16-bit register select-source: 00 = BC, 01 = DE, 10 = HL, 11 = SP
`v`	Restart address select: 010 = 0020h, 011 = 0030h, 100 = 0040h, 101 = 0050h, 111 = 0070h
`x`	8-bit constant to load into the XPC
`xx`	16-bit register select: 00 = BC, 01 = DE, 10 = IX, 11 = SP
`yy`	16-bit register select: 00 = BC, 01 = DE, 10 = IY, 11 = SP
`zz`	16-bit register select: 00 = BC, 01 = DE, 10 = HL, 11 = AF

a. The assembler translates a 16-bit constant or label to the 8-bit signed displacement.
b. The assembler translates a 16-bit constant or label to the 16-bit signed displacement.

1.6 Condition Codes

This section describes the condition codes you will see in Rabbit instructions or that are recognized by the Rabbit assembler.

Table 6: Condition Code Descriptions

Condition	Flag Bit Value	Description
NZ, NEQ	Z=0	The “Not Zero” or “Not Equal” condition is true if the result of the operation is not zero. By convention, NZ is used in conjunction with instructions like the BIT instruction and NEQ is more appropriate in conjunction with compare instructions.
Z, EQ	Z=1	The “Zero” or “Equal” condition is true if the result of the operation is zero. By convention, Z is used in conjunction with instructions like the BIT instruction and EQ is more appropriate in conjunction with compare instructions.
NC	C=0	The “No Carry” condition is true if the operation does not cause a carry.
C, LTU	C=1	The “Carry” condition is true if the operation causes a carry.
GT	(Z or (S xor V))=0	The “Greater Than” condition is true if the Z flag is zero and the L/V flag and the S flag are either both one or both zero.
LT	(S xor V)=1	The “Less Than” condition is true when the S flag is one and there is no arithmetic overflow (L/V=0); or the S flag is zero and there is arithmetic overflow (L/V=1).
GTU	((C=0) and (Z=0))=1	The “Greater Than Unsigned” condition is true if the C flag and Z flag are both zero.
P	S=0	The “Positive” condition is true if the S flag is zero.
M	S=1	The “Minus” condition is true if the S flag is one.
LZ	L/V=0	The “Logic Zero” condition is true if all of the four most significant bits of the operation’s result are zero.
LO	L/V=1	The “Logic One” condition is true if one or more of the four most significant bits of the operation’s result are one.
NV	L/V=0	The “No Overflow” condition is true if the arithmentic operation causes no overflow
V	L/V=1	The “Overflow” condition is true if the arithmentic operation causes an overflow

1.7 Definitions

This section defines some symbols, terms and representations that are used in this manual.

@PC

16-bit constant for the current code location.

CF

Represents the carry flag. The letter “C” also represents the carry flag, but only in the table heading that describes the bits in the flags register; otherwise, it represents the 8-bit Rabbit register.

Arithmetic Overflow

An arithmetic overflow happens when the result of an arithmetic operation is larger than the register or memory location in which it is stored. The Rabbit sets the overflow flag “V” when this happens.

Atomic

Describes an operation that is indivisible. It must happen completely or not at all. All Rabbit instructions are atomic except for the move instructions (LDDR, etc.) that are interruptible between iterations. Some are “chained-atomic,” meaning that the instruction’s atomicity is extended to the instruction immediately following it.

Little Endian

This is the byte-ordering method used by the Rabbit microprocessor. Numbers are stored low-byte first. You will see evidence of this in the opcode of instructions that take a multi-byte value; e.g., the opcode for the instruction “JP mn” is “C3 n m” where the low-byte of the 16-bit constant comes before the high-byte.

Long Logical Address

This is a 32-bit address (0xFFFFxxxx) that is treated as a logical address, i.e., it is passed through the MMU for translation. The upper 16 bits are all ones. Only the lower 16 bits are significant.

Shift Operations

The Rabbit has shift left and shift right instructions. Most of the shift instructions work on bits, but some (RLA and RRA) work at the byte level. Basically, a bit-level left shift accomplishes a multiply by 2 and a bit-level right shift does integer division by 2.

Bitwise shift operations are further distinquished by logical and arithmetic variations. For left shifts, there is no functional difference between a logical and arithmetic shift. However, for right shifts there is a difference: logical right shifts shift in a zero to the high-order bit, whereas for arithmetic right shifts, the high-bit is sign-extended.

Signed and Unsigned

Signed numbers can be either positive or negative. The high bit is the sign bit. A “1” means the number is negative; a “0” means it is positive.

Unsigned numbers are always positive. The benefit of using unsigned is that it doubles the number of unique positive numbers available.

Two’s Complement

Integer representation method that makes the circuitry for addition and subtraction less complex. The Rabbit uses two’s complement. This means that all negative integers have a “1” in their high bit. For example, let’s say the integer is -2. To find its two’s complement representation you take the binary representation of 2, then invert all the bits and add one. The binary representation of 2 is: 0000 0010; inverting the bits gives: 1111 1101; and adding one: 1111 1110; which is 0xFE in hex. So, 0xFE is the two’s complement representation of -2.

Chapter 2. Rabbit Processor Registers

The registers of the Rabbit family of microprocessors can be divided into two categories: processor registers and I/O registers. That last category can be divided further: parallel port registers, serial port registers, memory control registers, timer registers, etc. Information on I/O registers can be found in the Rabbit chip manuals and in the Dynamic C help file, accessible by selecting “I/O Registers” from the help menu.

The registers discussed in this chapter are the processor registers. These are the registers that are used in the Rabbit instruction set.

2.1 Rabbit 2000/3000 Processor Registers

The Rabbit 2000 and Rabbit 3000 microprocessors have identical processor register sets. The following table provides details.

Table 1. Rabbit 2000/3000 Register Set

Registers	8-Bit	16-Bit	Alternate Register
Accumulators	A	HL	A´, HL´
Flags[a]	F		F’
General Purpose	B, C, D, E, H, L	BC, DE	B´, C´, D´, E´, H´, L´ BC´, DE´
Index		IX, IY	None
Stack Pointer		SP	None
Program Counter		PC	None
Xmem Program Counter	XPC		None
Interrupt Priority	IP		None
Internal Interrupt	IIR		None
External Interrupt	EIR		None

a. S=Sign, Z=Zero, LV=Logical/Overflow, C=Carry. Bits marked “x” are reserved for future use.

Flag Bits

7							0
S	Z	x	x	x	L/V	x	C

2.2 Rabbit 4000/5000 Processor Registers

The Rabbit 4000 and Rabbit 5000 microprocessors have identical processor register sets. The following table provides details.The Rabbit 4000 and 5000 have an expanded register set compared to the Rabbit 2000 and 3000.

.

Table 2. Rabbit 4000/5000 Register Set

Registers	8-Bit	16-Bit	32-Bit	Alternate Registers
Accumulators	A	HL		A´, HL´
Flags[a]	F			F´
General Purpose	B, C, D, E, H, L	BC, DE, JK	BCDE, JKHL	B´, C´, D´, E´, H´, L´ BC´, DE´, JK´ BCDE´, JKHL´
Index		IX, IY	PW, PX, PY, PZ	PW´, PX´, PY´, PZ´
Stack Pointer		SP		None
Program Counter		PC		None
Xmem Program Counter	XPC	XPC (low 12 bits valid)		None
Interrupt Priority	IP			None
Internal Interrupt	IIR	SP		None
External Interrupt	EIR	PC		None
System/User Mode	SU			None
Handle Table Register	HTR			None

a. S=Sign, Z=Zero, LV=Logical/Overflow, C=Carry. Bits marked “x” are reserved for future use.

Flag Bits

7							0
S	Z	x	x	x	L/V	x	C

Chapter 3. OpCode Descriptions

This chapter includes complete descriptions for all Rabbit processor instructions. The instructions are listed alphabetically, with any Rabbit 2000/3000 instructions preceding their Rabbit 4000/5000 counterparts.

ADC A,*n*

Opcode	Instruction	Operation
CE *n*	ADC A,*n*	A = A + *n* + CF

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•			

Description

The 8-bit constant *n* is summed with the C flag and A. The sum is stored in A.

The Rabbit 4000/5000 assemblers view “ADC A,n” and “ADC n” as equivalent instructions. In the latter case, A is used even though it is not explicitly stated.

ADC A,*r*

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000	2	n/a	n/a

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•			

Description

A is summed with the C flag and with *r* (any of the 8-bit registers A, B, C, D, E, H, or L). The result is stored in A.

The opcodes for these instructions are different than the same instructions in the Rabbit 4000/5000.

Add With Carry

`ADC A,r`

Opcode	Instruction	Operation
——	**ADC A,*r***	**A = A + *r* + CF**
7F 8F	ADC A,A	A = A + A + CF
7F 88	ADC A,B	A = A + B + CF
7F 89	ADC A,C	A = A + C + CF
7F 8A	ADC A,D	A = A + D + CF
7F 8B	ADC A,E	A = A + E + CF
7F 8C	ADC A,H	A = A + H + CF
7F 8D	ADC A,L	A = A + L + CF

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•			

Description

A is summed with the C flag and with *r* (any of the registers A, B, C, D, E, H, or L). The result is stored in A. The Rabbit 4000/5000 assemblers view “ADC A,r” and “ADC r” as equivalent instructions. In the latter case, A is used even though it is not explicitly stated.

The opcodes for these instructions are different than the same instructions in the Rabbit 2000, 3000 and 3000A.

`ADC A,(HL)`

Opcode	Instruction	Operation
8E	ADC A,(HL)	A = A + (HL) + CF

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000	5	n/a	n/a

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•		•	

Description

A is summed with the C flag and with the data whose address is in HL.The result is stored in A.

The opcode for this instruction is different than the same instruction in the Rabbit 4000 and 5000.

ADC A,(HL)

Opcode	Instruction	Operation
7F 8E	ADC A,(HL)	A = A + (HL) + CF

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	7	n/a	n/a
Rabbit 5000	5	5	5

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•		•	

Description

The data in A is summed with the C flag and with the data whose address is in HL.The result is stored in A. The Rabbit 4000/5000 assemblers view "ADC A,(HL)" and "ADC (HL)" as equivalent instructions. In the latter case, A is used even though it is not explicitly stated.

The opcode for this instruction is different than the same instruction in the Rabbit 2000, 3000 and 3000A.

`ADC A,(IX+d)`
`ADC A,(IY+d)`

Opcode	Instruction	Operation
DD 8E *d*	ADC A,(IX+*d*)	A = A + (IX+*d*) + CF
FD 8E *d*	ADC A,(IY+*d*)	A = A + (IY+*d*) + CF

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	9	n/a	n/a
Rabbit 5000	10	9	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•		•	

Description

A is summed with the C flag and with the data whose address is:

- the sum of IX and the 8-bit signed displacement value *d*, or
- the sum of IY and the 8-bit signed displacement value *d*.

The result is stored in A.

The Rabbit 4000/5000 assemblers view "ADC A,(IX+d)" and "ADC (IX+d)" as equivalent instructions. In the latter case, A is used even though it is not explicitly stated. The same is true for "ADC A,(IY+d)" and "ADC (IY+d)."

ADC HL,*ss*

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•			

Description

HL is summed with the C flag and with *ss* (any of BC, DE, HL, or SP). The result is stored in HL.

ADD A,*n*

Opcode	Instruction	Operation
C6 *n*	ADD A,*n*	A = A + *n*

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•			

Description

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

The Rabbit 4000/5000 assemblers view “ADD A,n” and “ADD n” as equivalent instructions. In the latter case, A is used even though it is not explicitly stated.

ADD A,*r*

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000	2	n/a	n/a

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•			

Description

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

The opcodes for these instructions are different than the same instructions in the Rabbit 4000 and 5000.

ADD A,*r*

Opcode	Instruction	Operation
——	**ADD A,*r***	**A = A + *r***
7F 87	ADD A,A	A = A + A
7F 80	ADD A,B	A = A + B
7F 81	ADD A,C	A = A + C
7F 82	ADD A,D	A = A + D
7F 83	ADD A,E	A = A + E
7F 84	ADD A,H	A = A + H
7F 85	ADD A,L	A = A + L

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•			

Description

A is summed with *r* (any of the registers A, B, C, D, E, H, or L). The result is stored in A.The Rabbit 4000/5000 assemblers view “ADD A,r” and “ADD r” as equivalent instructions. In the latter case, A is used even though it is not explicitly stated.

The opcodes for these instructions are different than the same instructions in the Rabbit 2000, 3000 and 3000A.

ADD A,(HL)

Opcode	Instruction	Operation
86	ADD A,(HL)	A = A + (HL)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000	5	n/a	n/a

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•		•	

Description

A is summed with the data whose address is in HL. The result is stored in A.

The opcode for this instruction is different than the same instruction in the Rabbit 4000 and 5000.

ADD A,(HL)

Opcode	Instruction	Operation
7F 86	ADD A,(HL)	A = A + (HL)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	7	n/a	n/a
Rabbit 5000	5	5	5

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•		•	

Description

A is summed with the data whose address is in HL. The result is stored in A. The Rabbit 4000/5000 assemblers view "ADD A,(HL)" and "ADC (HL)" as equivalent instructions. In the latter case, A is used even though it is not explicitly stated.

The opcode for this instruction is different than the same instruction in the Rabbit 2000, 3000 and 3000A.

Add Without Carry 2000, 3000, 4000, 5000

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

Opcode	Instruction	Operation
DD 86 *d*	ADD A,(IX+*d*)	A = A + (IX+*d*)
FD 86 *d*	ADD A,(IY+*d*)	A = A + (IY+*d*)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	9	n/a	n/a
Rabbit 5000	10	9	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•		•	

Description

A is summed with the data whose address is:

- the sum of IX and the 8-bit signed displacement value *d*, or
- the sum of IY and the 8-bit signed displacement value *d*

The result is stored in A.

The Rabbit 4000/5000 assemblers view “ADD A,(IX+d)” and “ADD (IX+d)” as equivalent instructions. In the latter case, A is used even though it is not explicitly stated. The same is true for “ADD A,(IY+d)” and “ADD (IY+d).”

Add Without Carry

`ADD HL,JK`

Opcode	Instruction	Operation
65	ADD HL,JK	HL = HL + JK

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	•	•	•			

Description

HL is summed with JK. The result is stored in HL.

ADD HL,*ss*

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	•	•	•			

Description

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

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

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	•	•				

Description

IX or IY is summed with either itself or any of the registers BC, DE or SP. The result is stored in IX or IY.

ADD JKHL,BCDE

Opcode	Instruction	Operation
ED C6	ADD JKHL,BCDE	JKHL = JKHL + BCDE

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	•	•	•			

Description

JKHL is summed with BCDE. The result is stored in JKHL.

`ADD SP,d`

Opcode	Instruction	Operation
27 *d*	ADD SP,*d*	SP = SP + *d*

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	•	•				

Description

SP is summed with the 8-bit signed displacement *d*. The result is stored in SP.

ALTD

Opcode	Instruction	Operation
76	ALTD	Sets alternate register destination for following instruction.

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

This instruction prefix causes the instruction immediately following it to affect:

- the alternate flags, which is signified by "•" in the "F" column in the table above; or
- the alternate registers for the destination of the data, signified by "•" in the "R" column; or
- both of the above; or
- the instruction is not affected.

ALTD also causes special alternate register uses that are unique to some instructions (signified by "•" in the "SP" column). The instructions are:

```
EX BC,HL
EX DE,HL
EX JK',HL
EX BC',HL
EX DE',HL
```

How ALTD affects an instruction is noted in the Flags table for that instruction.

Example

The instruction: "ALTD ADD HL,DE" would add DE to HL and store the result in the alternate register HL' instead of in HL. In the information for "ADD HL,DE" both the columns "F" and "R" are marked, meaning that not only is the alternate register used, but so is the alternate flag.

The instructions "ALTD LD DE,BC" and "LD DE',BC" both load the data in BC into the alternate register DE' because the Dynamic C assembler recognizes them as the same instruction.

AND HL,DE

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•			

Description

Performs a bitwise AND operation between the word in HL and the word in DE. The result is stored in HL.

AND IX,DE
AND IY,DE

Opcode	Instruction	Operation
DD DC	AND IX,DE	IX = IX & DE
FD DC	AND IY,DE	IY = IY & DE

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•				

Description

Performs a bitwise AND operation between IX or IY and DE. The result is stored in IX or IY.

AND JKHL,BCDE

Opcode	Instruction	Operation
ED E6	AND JKHL,BCDE	JKHL = JKHL & BCDE

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•			

Description

Performs a bitwise AND operation between the 32-bit registers JKHL and BCDE. The result is stored in JKHL.

AND *n*

Opcode	Instruction	Operation
E6 *n*	AND *n*	A = A & *n*

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•			

Description

Performs a bitwise AND operation between A and the 8-bit constant *n*. The result is stored in A.

The Rabbit 4000/5000 assemblers view “AND A,n” and “AND n” as equivalent instructions.

AND *r*

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000	2	n/a	n/a

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•			

Description

Performs a bitwise AND operation between A and *r* (any of the registers A, B, C, D, E, H, or L). The result is stored in A.

The opcodes for these instructions are different than the same instructions in the Rabbit 4000 and 5000.

`AND r`

Opcode	Instruction	Operation
——	**AND *r***	**A = A & *r***
7F A7	AND A	A = A & A
7F A0	AND B	A = A & B
7F A1	AND C	A = A & C
7F A2	AND D	A = A & D
7F A3	AND E	A = A & E
7F A4	AND H	A = A & H
7F A5	AND L	A = A & L

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•			

Description

Performs a bitwise AND operation between A and *r* (any of the registers A, B, C, D, E, H, or L). The result is stored in A. The Rabbit 4000/5000 assemblers view “AND A,r” and “AND r” as equivalent instructions.

The opcodes for these instructions are different than the same instructions in the Rabbit 2000, 3000 and 3000A.

AND (HL)

Opcode	Instruction	Operation
A6	AND (HL)	A = A & (HL)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000	5	n/a	n/a

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•		•	

Description

Performs a bitwise AND operation between A and the byte whose address is in HL. The result is stored in A.

The opcode for this instruction is different than the same instruction in the Rabbit 4000 and 5000.

Example

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

```
AND (HL)
```

would result in the byte in A becoming 1001 0100.

Bitwise AND

AND (HL)

Opcode	Instruction	Operation
7F A6	AND (HL)	A = A & (HL)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	5	n/a	n/a
Rabbit 5000	7	7	5

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•		•	

Description

Bitwise AND operation between A and the byte whose address is in HL. The result is stored in A.The Rabbit 4000/5000 assemblers view “AND A,(HL)” and “AND (HL)” as equivalent instructions.

The opcode for this instruction is different than the same instruction in the Rabbit 2000, 3000 and 3000A.

Example

If the byte in A contains the value 1011 1100 and the byte at the memory location addressed in 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 (IX+*d*)
AND (IY+*d*)

Opcode	Instruction	Operation
DD A6 *d*	AND (IX+*d*)	A = A & (IX+*d*)
FD A6 *d*	AND (IY+*d*)	A = A & (IY+*d*)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	9	n/a	n/a
Rabbit 5000	10	9	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•		•	

Description

Performs a bitwise AND operation between A and the byte whose address is:

- the sum of IX and the 8-bit signed displacement *d*, or
- the sum of IY and the 8-bit signed displacement *d*

The result is stored in A.

The Rabbit 4000/5000 assemblers view “AND A,(IX+d)” and “AND (IX+d)” as equivalent instructions. The same is true for “AND A,(IY+d)” and “AND (IY+d).”

Example

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

```
AND (IX+d)
```

would result in the byte in A becoming 1001 0100.

Bit Test

BIT *b*,*r*

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	•	-	-	•				

Description

Tests bit *b* (any of the bits 0, 1, 2, 3, 4, 5, 6, or 7) of *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.

BIT *b*,(HL)

Opcode	Instruction	Operation
——	**BIT *b*,(HL)**	**(HL) & bit**
CB 46	BIT 0,(HL)	(HL) & bit 0
CB 4E	BIT 1,(HL)	(HL) & bit 1
CB 56	BIT 2,(HL)	(HL) & bit 2
CB 5E	BIT 3,(HL)	(HL) & bit 3
CB 66	BIT 4,(HL)	(HL) & bit 4
CB 6E	BIT 5,(HL)	(HL) & bit 5
CB 76	BIT 6,(HL)	(HL) & bit 6
CB 7E	BIT 7,(HL)	(HL) & bit 7

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	7	n/a	n/a
Rabbit 5000	7	7	5

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	•	-	-				•	

Description

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

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

This is a chained-atomic instruction, meaning that an interrupt cannot take place between this instruction and the instruction following it.

Bit Test

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

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	10	n/a	n/a
Rabbit 5000	11	9	7

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	•	-	-	•			•	

Description

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

- the sum of data in IX plus the 8-bit signed displacement value *d*, or
- the sum of data in IY plus the 8-bit signed displacement value *d*.

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

BOOL HL

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	0	0	•	•			

Description

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

BOOL IX
BOOL IY

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	0	0					

Description

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

`CALL mn`

Opcode	Instruction	Operation

CD *n m*	CALL *mn*	(SP - 1) = PC_{high}
		(SP - 2) = PC_{low}
		PC = *mn*
		SP = SP - 2

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	12	n/a	n/a
Rabbit 5000	13	11	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

This instruction is used to call a subroutine. First 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*, which is the16-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 the instruction

```
CALL label
```

where *mn* is coded as a `label`.

Call Subroutine

CALL (HL)
CALL (IX)
CALL (IY)

Opcode	Instruction	Operation
ED EA	CALL (HL)	(SP - 1) = PC_{high} (SP - 2) = PC_{low} PC = HL; SP = SP - 2
DD EA	CALL (IX)	(SP - 1) = PC_{high} (SP - 2) = PC_{low} PC = IX; SP = SP - 2
FD EA	CALL (IY)	(SP - 1) = PC_{high} (SP - 2) = PC_{low} PC = IY; SP = SP - 2

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	12	n/a	n/a
Rabbit 5000	13	13	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

This instruction is used to call a subroutine. First 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 the value in HL, IX or IY, the 16-bit address of the first instruction of the subroutine. SP is updated to reflect the two bytes pushed onto the stack.

Change Bits Under Mask

`CBM n`

Opcode	Instruction	Operation
ED 00 *n*	CBM *n*	tmp = [(HL) & ~n] | [A & n] (HL) = tmp (DE) = tmp

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	15	n/a	n/a
Rabbit 5000	15	14	13

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					•

Description

This instruction sets specified bits in an I/O register, where:

A = requested bits to be set in I/O register

n = 8-bit mask identifies bits that can be changed

DE = address of I/O register

HL = address of shadow register for I/O register

A bitwise AND operation is performed on the value in the shadow register and the inverse of the bitmask; which results in preserving any bits already set in the I/O register that are not under the bitmask. A second bitwise AND operation is performed on A and the bitmask; which results in setting all bits that are both requested (A) and allowed (n). The results of the two AND operations are then bitwise OR'd. This final answer is saved first in the shadow register and then in the I/O register.

Only (DE) is affected by IOI or IOE.

Change Carry Flag 2000, 3000, 4000, 5000

`CCF`

Opcode	Instruction	Operation
3F	CCF	CF = ~CF

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	•	•				

Description

The C flag is inverted. If it is set, it becomes cleared. If it is not set, it becomes set.

Clear <span>4000, 5000</span>

CLR HL

Opcode	Instruction	Operation
BF	CLR HL	HL = 0

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

HL is set to 0.

Convert Code Address 4000, 5000

CONVC *pp*

Opcode	Instruction	Operation
ED *pp*	**CONVC *pp***	**Convert *pp* to physical code address**
ED 0E	CONVC PW	Convert PW to physical address
ED 1E	CONVC PX	Convert PX to physical address
ED 2E	CONVC PY	Convert PY to physical address
ED 3E	CONVC PZ	Convert PZ to physical address

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	8	n/a	n/a
Rabbit 5000	8	8	6

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Converts the 16-bit logical address in the low word of *pp* (one of the 32-bit registers PW, PX, PY or PZ) to a 24-bit physical device offset, which replaces the logical address stored in *pp*. The actual number of bits used for the physical device offset depends on the available memory.

Convert Data Address

4000, 5000

CONVD pp

Opcode	Instruction	Operation
ED *pp*	**CONVD *pp***	**Convert *pp* to physical data address**
ED 0F	CONVD PW	Convert PW to physical address
ED 1F	CONVD PX	Convert PX to physical address
ED 2F	CONVD PY	Convert PY to physical address
ED 3F	CONVD PZ	Convert PZ to physical address

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	8	n/a	n/a
Rabbit 5000	8	8	6

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Converts the 16-bit logical address in the low word of *pp* (one of the 32-bit registers PW, PX, PY or PZ) to a 24-bit physical device offset, which replaces the logical address stored in *pp*. The actual number of bits used for the physical device offset depends on the available memory.

Block Copy

COPY

Opcode	Instruction	Operation

ED 80	COPY	(PY) = (PX)
		BC = BC - 1
		PY = PY + 1
		PX = PX + 1
		repeat while {BC != 0}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	n/a	7+7i[a]	n/a
Rabbit 5000	5+7i[1]	7+7i[1]	7+7i[1]

a. "i" is the number of bytes copied

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	•	-					

Description

This is a physical address block copy operation. It copies the number of bytes specified in BC starting from the address in PX to the address in PY, incrementing PY and PX for each successive byte.

Putting a physical address in the index registers means that the memory management unit (MMU) is not used by this instruction. Also, interrupts are possible between loops.

COPYR

Opcode	Instruction	Operation

ED 88	COPYR	(PY) = (PX)
		BC = BC - 1
		PY = PY - 1
		PX = PX - 1
		repeat while {BC != 0}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	n/a	7+7i[a]	n/a
Rabbit 5000	5+7i[1]	7+7i[1]	7+7i[1]

a. “i” is the number of bytes copied

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

This is a physical address block copy operation. It copies the number of bytes specified in BC starting from the address in PX to the address in PY, decrementing PY and PX for each successive byte.

Putting a physical address in the index registers means that the memory management unit (MMU) is not used by this instruction. Also, interrupts are possible between loops.

Compare **4000, 5000**

`CP HL,`*`d`*

Opcode	Instruction	Operation
48 *d*	CP HL,*d*	HL - *d* (d sign-extended to 16 bits)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•				

Description

Compares HL with the 8-bit signed value *d*, which is sign-extended to 16 bits. These compares are accomplished by subtracting *d* from HL. The result is:

```
HL < d : S=1, C=1, Z=0, L/V=V
HL = d : S=0, C=0, Z=1, L/V=V
HL > d : S=0, C=0, Z=0, L/V=V
```

where "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 (HL in this case). For example, the overflow flag will be set if HL contains 0x8000 and you're comparing it to 0x01 (sign-extended to 0x0001). The result of the subtraction is 0x7FFF, which has a different sign than 0x8000.

This operation does not affect HL.

CP HL,DE

Opcode	Instruction	Operation
ED 48	CP HL,DE	HL - DE

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•				

Description

Compares HL with DE. These compares are accomplished by subtracting DE from HL. The result is:

```
HL < DE : S=1, C=1, Z=0, L/V=V
HL = DE : S=0, C=0, Z=1, L/V=V
HL > DE : S=0, C=0, Z=0, L/V=V
```

where “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 (HL in this case). For example, the overflow flag will be set after the compare instruction if HL contains 0x8000 and DE contains 0x0001. The result of the subtraction is 0x7FFF, which has a different sign than 0x8000.

This operation does not affect HL or DE.

Compare 4000, 5000

CP JKHL,BCDE

Opcode	Instruction	Operation
ED 58	CP JKHL,BCDE	JKHL - BCDE

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•				

Description

Compares JKHL with BCDE. These compares are accomplished by subtracting BCDE from JKHL. The result is:

```
BCDE > JKHL : S=1, C=1, Z=0, L/V=V
BCDE = JKHL : S=0, C=0, Z=1, L/V=V
BCDE < JKHL : S=0, C=0, Z=0, L/V=V
```

where “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 (JKHL in this case). For example, the overflow flag will be set after the compare instruction if JKHL contains 0x80000000 and BCDE contains 0x00000001. The result of the subtraction is 0x7FFFFFFF, which has a different sign than 0x80000000.

This operation does not affect JKHL or BCDE.

CP *n*

Opcode	Instruction	Operation
FE *n*	CP *n*	A - *n*

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•				

Description

Compares 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
```

where "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, the overflow flag will be set if A contains 0x80 and you are comparing it to 0x01. The result of the subtraction is 0x7F, which has a different sign than 0x80.

The Rabbit 4000/5000 assemblers view "CP A,n" and "CP n" as equivalent instructions.

This operation does not affect A.

Compare

CP *r*

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000	2	n/a	n/a

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•				

Description

Compares A with *r* (any of the registers A, B, C, D, E, H, or L). This compare is accomplished by subtracting *r* from A. The result is:

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

where “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, the overflow flag will be set if A contains 0x80 and you're comparing it to 0x01. The result of the subtraction is 0x7F, which has a different sign than 0x80.

This operation does not affect A.

The opcode for this instruction is different than the same instruction in the Rabbit 4000 and 5000.

CP *r*

Opcode	Instruction	Operation
——	**CP *r***	**A - *r***
7F BF	CP A	A - A
7F B8	CP B	A - B
7F B9	CP C	A - C
7F BA	CP D	A - D
7F BB	CP E	A - E
7F BC	CP H	A - H
7F BD	CP L	A - L

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•				

Description

Compares A with *r* (any of the registers A, B, C, D, E, H, or L). This compare is accomplished by subtracting *r* from A. The result is:

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

where "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, the overflow flag will be set if A contains 0x80 and you are comparing it to 0x01. The result of the subtraction is 0x7F, which has a different sign than 0x80.

The Rabbit 4000/5000 assemblers view "CP A,r" and "CP r" as equivalent instructions.

This operation does not affect A or *r*.

The opcode for this instruction is different than the same instruction in the Rabbit 2000, 3000 and 3000A.

CP (HL)

Opcode	Instruction	Operation
BE	CP (HL)	A - (HL)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000	5	n/a	n/a

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•			•	

Description

Compares A with the data whose address is in HL.

These compares are accomplished by subtracting the data from A. (This operation does not affect 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 addressed data 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, the overflow flag will be set if A contains 0x80 and you're comparing it to 0x01. The result of the subtraction is 0x7F, which has a different sign than 0x800.

The opcode for this instruction is different than the same instruction in the Rabbit 4000 and 5000.

Compare

`CP (HL)`

Opcode	Instruction	Operation
7F BE	CP (HL)	A - (HL)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	7	n/a	n/a
Rabbit 5000	7	7	5

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•			•	

Description

Compares A with the data whose address is in HL. These compares are accomplished by subtracting the data from A. (This operation does not affect 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 addressed data 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, the overflow flag will be set if A contains 0x80 and you are comparing it to 0x01. The result of the subtraction is 0x7F, which has a different sign than 0x80.

The Rabbit 4000/5000 assemblers view “CP A,(HL)” and “CP (HL)” as equivalent instructions.

The opcode for this instruction is different than the same instruction in the Rabbit 2000, 3000 and 3000A.

Compare **2000, 3000, 4000, 5000**

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

Opcode	Instruction	Operation
DD BE *d*	CP (IX+*d*)	A - (IX+*d*)
FD BE *d*	CP (IY+*d*)	A - (HL+*d*)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	9	n/a	n/a
Rabbit 5000	10	9	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•			•	

Description

Compares A with the data whose address is:

• the sum of IX and the 8-bit signed displacement value *d*, or

• the sum of IY and the 8-bit signed displacement value *d*.

These compares are accomplished by subtracting the data from A. (This operation does not affect 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 addressed data 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, the overflow flag will be set if A contains 0x80 and you are comparing it to 0x01. The result of the subtraction is 0x7F, which has a different sign than 0x800.

The Rabbit 4000/5000 assemblers view “CP A,(IX+d)” and “CP (IX+d)” as equivalent instructions. The same is true for “CP A,(IY+d)” and “CP (IY+d).”

CPL

Opcode	Instruction	Operation
2F	CPL	A = ~A

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

A is inverted (one’s complement).

Example

If A is 1100 0101, it will be 0011 1010 after the CPL instruction executes.

Decrement 2000, 3000, 4000, 5000

DEC IX
DEC IY

Opcode	Instruction	Operation
DD 2B	DEC IX	IX = IX - 1
FD 2B	DEC IY	IY = IY - 1

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Decrements IX or IY.

DEC *r*

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	-	•	•			

Description

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

DEC *ss*

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Decrements *ss* (any of BC, DE, HL, or SP).

DEC (HL)

Opcode	Instruction	Operation
35	DEC (HL)	(HL) = (HL) - 1

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	8	n/a	n/a
Rabbit 5000	8	8	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	-	•			•	•

Description

Decrements the byte whose address is HL

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

Opcode	Instruction	Operation
DD 35 *d*	DEC (IX+D)	(IX+*d*) = (IX+*d*) - 1
FD 35 *d*	DEC (IY+D)	(IY+*d*) = (IY+*d*) - 1

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	12	n/a	n/a
Rabbit 5000	13	12	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	-	•			•	•

Description

Decrements the byte whose address is:

- the sum of IX and the 8-bit signed displacement value *d*, or
- the sum of IY and the 8-bit signed displacement value *d*.

Decrement and Jump if Not Zero

2000, 3000, 4000, 5000

`DJNZ` *`label`*

Opcode	Instruction	Operation

10 *e*	DJNZ *label*
	DJNZ *mn*[a]
		B = B - 1
		if {B != 0} then PC = PC[b] + *e*

a. The 16-bit constant *mn* is the destination logical address of the jump.
b. The value of PC after the instruction fetch.

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	5	n/a	n/a
Rabbit 5000	6	6	6

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

This instruction controls program flow by allowing conditional jumps to specified locations.

First, B is decremented. If B does not equal zero, the instruction transfers control to the specified address. The address is specified by a label or logical address. The assembler translates the label or logical address "mn" to an 8-bit signed displacement value "e".

The displacement value "e" is relative to the address of the first byte of the instruction following DJNZ. This fact is because the processor calculates the new PC value after it increments the PC for the instruction fetch of DJNZ.

If B does equal zero, PC is incremented normally.

Note that the relative jump has a limited range of [-128, 127] from the address of the first byte of the instruction following the DJNZ instruction.

Decrement Word and Jump if Not Zero 4000, 5000

`DWJNZ` *`label`*

Opcode	Instruction	Operation

ED 10 *e*	DWJNZ *label*
	DWJNZ *mn*[a]
		BC = BC - 1
		if {BC != 0} PC = PC[b] + *e*

a. The 16-bit constant *mn* is the destination logical address of the jump.
b. The value of PC after the instruction fetch.

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	7	n/a	n/a
Rabbit 5000	8	8	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

This instruction controls program flow by allowing conditional jumps to specified locations.

First, BC is decremented. If BC does not equal zero, the instruction transfers control to the specified address. The address is specified by a label or logical address. The assembler translates the label or logical address “mn” to an 8-bit signed displacement value “e”.

The displacement value “e” is relative to the address of the first byte of the instruction following DWJNZ. This fact is because the processor calculates the new PC value after it increments the PC for the instruction fetch of DWJNZ.

If BC does equal zero, PC is incremented normally.

Note that the relative jump has a limited range of [-128, 127] from the address of the first byte of the instruction following the DWJNZ instruction.

EX AF,AF'

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Exchanges AF with its alternate register, AF'.

Exchange 4000, 5000

`EX BC,HL`

Opcode	Instruction	Operation

B3	EX BC,HL	if (!ALTD) then BC <–> HL
		else BC <–> HL’

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	2	n/a	n/a
Rabbit 5000	13	12	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Exchanges BC with HL. If the instruction is preceded by ALTD, the alternate register HL' is used instead of HL.

EX BC',HL

Opcode	Instruction	Operation

ED 74	EX BC',HL	if (!ALTD) then BC' <–> HL
		else BC' <–> HL'

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	4

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Exchanges BC' with HL. If the instruction is preceded by ALTD, the alternate register HL' is used instead of HL.

Exchange 2000, 3000, 4000, 5000

`EX DE,HL`
`EX DE′,HL`

Opcode	Instruction	Operation

EB	EX DE,HL	if (!ALTD) then DE <–> HL
		else DE <–> HL'

E3	EX DE',HL	if (!ALTD) then DE' <–> HL
		else DE' <–> HL'

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-			•		

Description

Exchanges DE or DE′ with HL. If the instruction is preceded by ALTD, the alternate register HL' is used instead of 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’

EX JK,HL

Opcode	Instruction	Operation

B9	EX JK,HL	if (!ALTD) then JK <–> HL
		else JK <–> HL'

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-				•	

Description

Exchanges JK with HL. If the instruction is preceded by ALTD, the alternate register HL' is used instead of HL.

The Dynamic C assembler recognizes the following instruction, which is based on a combination of ALTD and the above exchange operation:

```
EX JK,HL'          ; equivalent to ALTD EX JK',HL'
```

EX JK',HL

Opcode	Instruction	Operation

ED 7C	EX JK',HL	if (!ALTD) then JK' <–> HL
		else JK' <–> HL’

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-				•	

Description

Exchanges JK' with HL. If the instruction is preceded by ALTD, the alternate register HL' is used instead of HL.

The Dynamic C assembler recognizes the following instruction, which is based on a combination of ALTD and the above exchange operation:

• `EX JK’,HL’` ; equivalent to ALTD EX JK',HL

EX JKHL,BCDE

Opcode	Instruction	Operation
B4	EX JKHL,BCDE	JKHL <–> BCDE

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Exchanges JKHL with BCDE.

EX (SP),HL

Opcode	Instruction	Operation
ED 54	EX (SP),HL	H <–> (SP+1) L <–> (SP)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	15	n/a	n/a
Rabbit 5000	15	15	13

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Exchanges the 16 bits at the top of the stack (whose address is SP) with HL.

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

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	15	n/a	n/a
Rabbit 5000	15	15	13

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Exchanges 16 bits at the top of the stack (whose address is SP) with IX or IY.

EXP

Opcode	Instruction	Operation
ED D9	EXP	PW <–> PW' PX <–> PX' PY <–> PY' PZ <–> PZ'

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
–	–	–	–					

Description

Exchanges PW, PX, PY and PZ with their respective alternate registers, PW', PX', PY' and PZ'.

EXX

Opcode	Instruction	Operation
D9	EXX	BC <–> BC' DE <–> DE' HL <–> HL' JK <–> JK' (Rabbit 4000/5000 only)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Exchanges BC, DE, and HL, with their respective alternate registers, BC', DE', and HL'.

If using the Rabbit 4000 or Rabbit 5000, this instructions also exchanges JK with its alternate JK'.

Check Condition Code

4000, 5000

`FLAG cc,HL`

Opcode	Instruction	Operation
—	**FLAG *cc*,HL**	**if (*cc*) then HL=1 else HL=0**
ED C4	FLAG NZ,HL	if (NZ) then HL=1 else HL=0
ED CC	FLAG Z,HL	if (Z) then HL=1 else HL=0
ED D4	FLAG NC,HL	if (NC) then HL=1 else HL=0
ED DC	FLAG C,HL	if (C) then HL=1 else HL=0
ED A4	FLAG GT,HL	if (GT) then HL=1 else HL=0
ED B4	FLAG LT,HL	if (LT) then HL=1 else HL=0
ED AC	FLAG GTU,HL	if (GTU) then HL=1 else HL=0
ED BC	FLAG V,HL	if (V) then HL=1 else HL=0

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

If the condition *cc* is true then HL is set to one. Otherwise, HL is reset to zero.

Condition Code	Flag Bit Value	Description
NZ	Z=0	True when Z flag has not been set
Z	Z=1	True when Z flag has been set
NC	C=0	True when C flag has not been set
C	C=1	True when the C flag has been set
GT	(Z or (S xor V))=0	True when Z is 0 and L/V and S are either both 1 or both 0.
LT	(S xor V)=1	True when either S or L/V is 1.
GTU	((C=0) and (Z=0)) =1	True when C and Z are both 0.
V	L/V=1	True when L/V flag is set: there is overflow.

Fast System Call

FSYSCALL

Opcode	Instruction	Operation

ED 55	FSYSCALL	(SP - 1) = PC_{high}
		(SP - 2) = PC_{low}
		(SP - 3) = SU
		SP = SP - 3
		PC = {IIR,0x60}
		SU = {SU[5:0],00}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	15	n/a	n/a
Rabbit 5000	16	16	14

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	1	•				

Description

Pushes PC and SU on the stack. SU is set to system mode and PC is set 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 0x100 boundary.

FSYSCALL is essentially a new RST opcode, added to allow access to system space without using one of the existing RST opcodes. It will put the processor into System mode and execute code in the corresponding interrupt-vector table entry.

Inverse SBOX <span>4000, 5000</span>

`IBOX A`

Opcode	Instruction	Operation
ED 12	IBOX A	A = ibox(A)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

The inverse sbox structure is a 256-byte lookup table used by the AES-128 cipher. A contains the index into the table and is replaced by the referenced value from the table.

IDET

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 3000A/4000	2	n/a	n/a
Rabbit 5000			

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

The IDET instruction asserts a System Mode Violation interrupt if System/User mode is enabled (which is done 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 the instruction “LD E,E” and actually executes that opcode as well as the behavior described above. If IDET is prefixed by ALTD, the instruction LD E’,E is executed and the special System/User mode behavior does not occur.

Increment

INC IX
INC IY

Opcode	Instruction	Operation
DD 23	INC IX	IX = IX + 1
FD 23	INC IY	IY = IY + 1

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Increments IX or IY.

INC *r*

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	-	•	•			

Description

Increments *r* (any of the 8-bit registers A, B, C, D, E, H, or L).

INC *ss*

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Increments *ss* (any of 16-bit registers BC, DE, HL, or SP).

INC (HL)

Opcode	Instruction	Operation
34	INC (HL)	(HL) = (HL) + 1

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	8	n/a	n/a
Rabbit 5000	8	8	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	-	•			•	•

Description

Increments the byte whose address is HL.

Increment

`INC (IX+d)`
`INC (IY+d)`

Opcode	Instruction	Operation
DD 34 *d*	INC (IX+*d*)	(IX+*d*) = (IX+*d*) + 1
FD 34 *d*	INC (IY+*d*)	(IY+*d*) = (IY+*d*) + 1

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	12	n/a	n/a
Rabbit 5000	13	12	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	-	•			•	•

Description

Increments the byte whose address is:

- the sum of IX and the 8-bit signed displacement *d*, or
- the sum of IY and the 8-bit signed displacement value *d*

Input/Output External/Internal 2000, 3000, 4000, 5000

IOE
IOI

Opcode	Instruction	Operation
D3	IOI	I/O internal prefix
DB	IOE	I/O external prefix

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

- `IOI:` The IOI prefix allows the use of existing memory access instructions as internal I/O instructions. Writes to internal I/O registers require two clocks rather than the three required for memory write operations.

  If an IOI prefix effects the destination of an instruction, that is, it causes an internal I/O write instead of a memory write, then the net effect of adding an IOI prefix to such an instruction is to add one cycle to the total time for the instruction because an internal write takes 2 cycles instead of 3, while the instruction fetch for the prefix byte adds 2 cycles.

  For Rabbit 2000 and 3000 only: 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. This does not apply to the Rabbit 3000A, 4000 or 5000.

- `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.

NOTE: If using the original Rabbit 2000 and a Dynamic C version prior to 6.57, read Technical Note 302 (TN302) for an easy solution to an unlikely problem.

Interrupt Priority Restore 2000, 3000, 4000, 5000

IPRES

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

The IPRES instruction rotates the contents of IP 2 bits to the right, replacing the current priority with the previous priority.

This is a chained-atomic instruction, meaning that an interrupt cannot take place between this instruction and the instruction following it.

Example

If IP contains 00000110, the execution of the instruction

```
IPRES
```

would cause IP to contain 10000001.

Interrupt Priority Set

IPSET 0
IPSET 1
IPSET 2
IPSET 3

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

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 are chained-atomic instructions, meaning that an interrupt cannot take place between one of these instructions and the instruction following it.

`IPSET 0`: shifts IP 2 bits to the left, then sets bits 0 and 1 of IP to 00
`IPSET 1`: shifts IP 2 bits to the left, then sets bits 0 and 1 of IP to 01
`IPSET 2`: shifts IP 2 bits to the left, then sets bits 0 and 1 of IP to 10
`IPSET 3`: shifts IP 2 bits to the left, then sets bits 0 and 1 of IP to 11

Processor Priority	Effect on Interrupts
0	All interrupts, priority 1,2 and 3, take place after execution of the current non chained-atomic instruction.
1	Only interrupts of priority 2 and 3 take place after execution of the current non chained-atomic instruction.
2	Only interrupts of priority 3 take place after execution of the current non chained-atomic instruction.
3	All interrupts are suppressed. Note that the RST instruction is not an interrupt.

Jump **4000, 5000**

`JP cx,mn`

Opcode	Instruction	Operation
—	**`JP cx,mn`**	**`if {cx} PC = mn`**
A2 *n m*	JP GT,mn	if {GT} PC = mn
B2 *n m*	JP LT,mn	if {LT} PC = mn
AA *n m*	JP GTU,mn	if {GTU} PC = mn
BA *n m*	JP V,mn	if {V} PC = mn

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	7	n/a	n/a
Rabbit 5000	7	5	5

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

If the condition *cx* is true then PC is loaded with the 16-bit constant *mn*. If the condition is false then PC increments normally.

Condition Code	Flag Bit Value	Description
GT	(Z or (S xor V))=0	True when Z is 0 and L/V and S are either both 1 or both 0.
LT	(S xor V)=1	True when either S is 1 or L/V is 1.
GTU	((C=0) and (Z=0))=1	True when C and Z are both 0.
V	L/V=1	True when the L/V flag is set: there is overflow.

JP *f*,*mn*

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	7	n/a	n/a
Rabbit 5000	7	5	5

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

If the condition *f* is true then PC is loaded with the 16-bit constant *mn*. If the condition is false then PC increments normally. The condition *f* is one of the following:

Condition Code	Flag Bit Value	Description
NZ	Z=0	True when Z flag has not been set
Z	Z=1	True when Z flag has been set
NC	C=0	True when C flag has not been set
C	C=1	True when C flag has been set
LZ	L/V=0	True when L/V flag has not been set
LO	L/V=1	True when L/V flag has been set
P	S=0	True when S flag has not been set
M	S=1	True when S flag has been set

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

`JP (HL)`

Opcode	Instruction	Operation
E9	JP (HL)	PC = HL

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	4

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

PC is loaded with HL. The Dynamic C assembler recognizes labels as well.

See Also: `SJP label`

Jump

`JP (IX)`
`JP (IY)`

Opcode	Instruction	Operation
DD E9	JP (IX)	PC = IX
FD E9	JP (IY)	PC = IY

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	6	n/a	n/a
Rabbit 5000	6	6	4

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

PC is loaded with IX or IY. The Dynamic C assembler recognizes labels as well.

See Also: `SJP label`

`JP` *`mn`*

Opcode	Instruction	Operation
C3 *n m*	JP *mn*	PC = *mn*

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	7	n/a	n/a
Rabbit 5000	7	5	5

Flags				ALTD			IOI/IOE	
`S`	`Z`	`L/V`	`C`	`F`	`R`	`SP`	`S`	`D`
-	-	-	-					

Description

PC is loaded with the 16-bit constant *mn*. The Dynamic C assembler recognizes labels as well.

See Also: `SJP label`

Jump Relative

`JR cc,label`

Opcode	Instruction	Operation
——	`JR cc,label` `JR cc,mn`[a]	**if {cc} PC = PC[b] + e**
20 *e*	JR NZ,*mn*	if {NZ} PC = PC + *e*
28 *e*	JR Z,*mn*	if {Z} PC = PC + *e*
30 *e*	JR NC,*mn*	if {NC} PC = PC + *e*
38 *e*	JR C,*mn*	if {C} PC = PC + *e*

a. The 16-bit constant *mn* is the destination logical address of the jump.
b. The value of PC after the instruction fetch.

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	5	n/a	n/a
Rabbit 5000	6	6	6

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

If condition *cc* is true, this instruction transfers control to the specifed address. The address is specified by a label or logical address. The assembler translates the label or logical address “mn” to an 8-bit signed displacement value, “e”.

The displacement value “e” is relative to the address of the first byte of the instruction following JR. This fact is because the processor calculates the new PC value after it increments the PC for the instruction fetch of JR. If condition *cc* is false, PC is incremented normally.

Condition Code	Flag Bit Value	Description
NZ	Z=0	True when the Z flag has not been set
Z	Z=1	True when the Z flag has been set
NC	C=0	True when the C flag has not been set
C	C=1	True when the C flag has been set

Note that the relative jump has a limited range of [-128, 127] from the address of the first byte of the instruction following the JR instruction.

JR *cx,label*

Opcode	Instruction	Operation
——	**JR cx,label** **JR cx,mn**[a]	**if {cx} PC = PC**[b] **+ e**
A0 *e*	JR GT,*mn*	if {GT} PC = PC + *e*
B0 *e*	JR LT,*mn*	if {LT} PC = PC + *e*
A8 *e*	JR GTU,*mn*	if {GTU} PC = PC + *e*
B8 *e*	JR V,*mn*	if {V} PC = PC + *e*

a. The 16-bit constant *mn* is the destination logical address of the jump.
b. The value of PC after the instruction fetch.

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	5	n/a	n/a
Rabbit 5000	6	6	6

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

If condition *cx* is true, this instruction transfers control to the specifed address. The address is specified by a label or logical address. The assembler translates the label or logical address “mn” to an 8-bit signed displacement value, “e”.

The displacement value “e” is relative to the address of the first byte of the instruction following JR. This fact is because the processor calculates the new PC value after it increments the PC for the instruction fetch of JR. If condition *cx* is false, PC is incremented normally.

Condition Code	Flag Bit Value	Description
GT	(Z or (S xor V))=0	True when Z is 0 and L/V and S are either both 1 or both 0.
LT	(S xor V)=1	True when either S or L/V is 1.
GTU	((C=0) and (Z=0))=1	True when C and Z are both 0.
V	L/V=1	True when L/V flag is set: there is overflow.

Note that the relative jump has a limited range of [-128, 127] from the address of the first byte of the instruction following the JR instruction.

JR *label*

Opcode	Instruction	Operation
18 *e*	JR label JR *mn*[a]	PC = PC[b] + *e*

a. The 16-bit constant *mn* is the destination logical address of the jump.
b. The value of PC after the instruction fetch.

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	5	n/a	n/a
Rabbit 5000	6	6	6

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

This instruction transfers control to the specifed address. The address is specified by a label or logical address. The assembler translates the label or logical address “mn” to an 8-bit signed displacement value, “e”.

The displacement value “e” is relative to the address of the first byte of the instruction following JR. This fact is because the processor calculates the new PC value after it increments the PC for the instruction fetch of JR.

Note that the relative jump has a limited range of [-128, 127] from the address of the first byte of the instruction following the JR instruction.

See Also: `SJP label`

Jump Relative 4000, 5000

`JRE cc,label`

Opcode	Instruction	Operation
——	**JRE cc,label** **JRE cc,mn**[a]	**if {cc} PC = PC**[b] **+ ee**
ED C3 *ee* $_{low}$ *ee*$_{high}$	JRE NZ,*mn*	if {NZ} PC = PC + *ee*
ED CB *ee* $_{low}$ *ee*$_{high}$	JRE Z,*mn*	if {Z} PC = PC + *ee*
ED D3 *ee* $_{low}$ *ee*$_{high}$	JRE NC,*mn*	if {NC} PC = PC + *ee*
ED DB *ee* $_{low}$ *ee*$_{high}$	JRE C,*mn*	if {C} PC = PC + *ee*

a. The 16-bit constant *mn* is the destination logical address of the jump.
b. The value of PC after the instruction fetch.

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	9	n/a	n/a
Rabbit 5000	10	10	10

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

If condition “cc” is true, this instruction transfers control to the specifed address. The address is specified by a label or logical address. The assembler translates the label or logical address “mn” to a 16-bit signed displacement value, “ee”.

The displacement value “ee” is relative to the address of the first byte of the instruction following JRE. This fact is because the processor calculates the new PC value after it increments the PC for the instruction fetch of JRE. If condition “cc” is not true, PC is incremented normally.

Condition Code	Flag Bit Value	Description
NZ	Z=0	True when Z flag has not been set
Z	Z=1	True when Z flag has been set
NC	C=0	True when C flag has not been set
C	C=1	True when C flag has been set

Note that the relative jump has a range of [-32768, 32767] from the address of the first byte of the instruction following the JRE instruction.

JRE *cx*,label

Opcode	Instruction	Operation
——	**JRE cx,label** **JRE cx,mn**[a]	**if {cx} PC = PC**[b] **+ ee**
ED A3 *ee* low *ee*high	JRE GT,*mn*	if {GT} PC = PC + *ee*
ED B3 *ee* low *ee*high	JRE LT,*mn*	if {LT} PC = PC + *ee*
ED AB *ee* low *ee*high	JRE GTU,*mn*	if {GTU} PC = PC + *ee*
ED BB *ee* low *ee*high	JRE V,*mn*	if {V} PC = PC + *ee*

a. The 16-bit constant *mn* is the destination logical address of the jump
b. The value of PC after the instruction fetch.

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	9	n/a	n/a
Rabbit 5000	10	10	10

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

If condition “cx” is true, this instruction transfers control to the specifed address. The address is specified by a label or logical address. The assembler translates the label or logical address “mn” to a 16-bit signed displacement value, “ee”.

The displacement value “ee” is relative to the address of the first byte of the instruction following JRE. This fact is because the processor calculates the new PC value after it increments the PC for the instruction fetch of JRE. If condition “cx” is not true, PC is incremented normally.

Condition Code	Flag Bit Value	Description
GT	(Z or (S xor V))=0	True when Z is 0 and L/V and S are either both 1 or both 0.
LT	(S xor V)=1	True when either S is 1 or L/V is 1.
GTU	((C=0) and (Z=0))=1	True when C and Z are both 0.
V	L/V=1	True when L/V flag is set: there is overflow.

Note that the relative jump has a range of [-32768, 32767] from the address of the first byte of the instruction following the JRE instruction.

Jump Relative 4000, 5000

JRE *label*

Opcode	Instruction	Operation
98 ee_{low} ee_{high}	JRE label JRE *mn*[a]	PC = PC^{b} + *ee*

a. The 16-bit constant *mn* is the destination logical address of the jump.
b. The value of PC after the instruction fetch.

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	7	n/a	n/a
Rabbit 5000	8	8	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

This instruction transfers control to the specifed address. The address is specified by a label or logical address. The assembler translates the label or logical address “mn” to a 16-bit signed displacement value, “ee”.

The displacement value “ee” is relative to the address of the first byte of the instruction following JRE. This fact is because the processor calculates the new PC value after it increments the PC for the instruction fetch of JRE.

The relative jump has a range of [-32768, 32767] from the address of the first byte of the instruction following the JRE instruction.

Long Call Subroutine

LCALL *x,mn*

Opcode	Instruction	Operation

CF *n m x*	LCALL *x,mn*	(SP - 1) = XPC
		(SP - 2) = PC_{high}
		(SP - 3) = PC_{low}
		XPC = *x*
		PC = *mn*
		SP = SP - 3

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	19	n/a	n/a
Rabbit 5000	20	18	16

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	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 constant *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 XPC) is dynamically changed with the LCALL instruction.

First, XPC is pushed on the stack. Next, PC is pushed on the stack, high-order byte first. Then XPC is loaded with the 8-bit constant *x* and PC is loaded with *mn*. The SP is updated to reflect the three bytes pushed onto it.

Alternate Forms

The Dynamic C assembler recognizes several forms of LCALL:

```
LCALL label
LCALL x:mn
LCALL x,mn
```

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

Note: Avoid mixing LCALL and LLCALL instructions. When LCALL pushes the XPC, it also clears the upper bits of the LXPC.

LD A,EIR
LD A,IIR

Opcode	Instruction	Operation
ED 57	LD A,EIR	A = EIR
ED 5F	LD A,IIR	A = IIR

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	–	–	•	•			

Description

Loads A with EIR or IIR.

EIR is used to specify the most significant byte of the External Interrupt address. The value loaded in EIR is concatenated with the appropriate External Interrupt address to form the 16-bit ISR starting address. IIR is used to specify the most significant byte of the Internal Peripheral Interrupt address. The value loaded in IIR is concatenated with the appropriate Internal Peripheral address to form the 16-bit ISR starting address for that peripheral.

LD A,HTR

Opcode	Instruction	Operation
ED 50	LD A,HTR	A = HTR

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads A with HTR.

LD A,XPC

Opcode	Instruction	Operation
ED 77	LD A,XPC	A = XPC

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads A with XPC.

This is a chained-atomic instruction, meaning that an interrupt cannot take place between this instruction and the instruction following it.

LD A,(BC)
LD A,(DE)

Opcode	Instruction	Operation
0A	LD A,(BC)	A = (BC)
1A	LD A,(DE)	A = (DE)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	6	n/a	n/a
Rabbit 5000	6	6	6

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•		•	

Description

Loads A with the data whose address is BC or DE.

`LD A,(mn)`

Opcode	Instruction	Operation
3A *n m*	LD A,(*mn*)	A = (*mn*)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	9	n/a	n/a
Rabbit 5000	9	7	7

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•		•	

Description

Loads A with the data whose address is the 16-bit constant *mn*.

LD A,(IX+A)
LD A,(IY+A)

Opcode	Instruction	Operation
DD 06	LD A,(IX+A)	A = (IX + A)
FD 06	LD A,(IY+A)	A = (IY + A)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	8	n/a	n/a
Rabbit 5000	9	9	7

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•		•	

Description

Loads A with the data whose address is:

- the sum of IX and A, or
- the sum of IY and A.

A is considered an 8-bit unsigned offset. These instructions are useful for accessing 256-byte lookup tables.

LD A,(*ps*+*d*)

Opcode	Instruction	Operation
—	**LD A,(*ps*+*d*)**	**A = (*ps* + *d*)**
8D *d*	LD A,(PW+*d*)	A = (PW + *d*)
9D *d*	LD A,(PX+*d*)	A = (PX + *d*)
AD *d*	LD A,(PY+*d*)	A = (PY + *d*)
BD *d*	LD A,(PZ+*d*)	A = (PZ + *d*)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	7	n/a	n/a
Rabbit 5000	8	8	7

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Load A with the data whose address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If *ps* is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address with only the low 16 bits being significant. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 or 24 bits (depending on the memory that is used) being significant.

The address is computed as the sum of *ps* and the 8-bit signed value *d*.

LD A,(*ps*+HL)

Opcode	Instruction	Operation
—	**LD A,(*ps*+HL)**	**A = (*ps* + HL)**
8B	LD A,(PW+HL)	A = (PW + HL)
9B	LD A,(PX+HL)	A = (PX + HL)
AB	LD A,(PY+HL)	A = (PY + HL)
BB	LD A,(PZ+HL)	A = (PZ + HL)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	6	n/a	n/a
Rabbit 5000	7	7	7

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Load A with the data whose address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If *ps* is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address, with only the low 16 bits being significant. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 or 24 bits (depending on the memory that is used) being significant.

The address is computed as the sum of *ps* and HL. HL is considered to be sign extended to 24 bits.

`LD BCDE,`***`ps`***

Opcode	Instruction	Operation
—	**LD BCDE,*ps***	**BCDE = ps**
DD CD	LD BCDE,PW	BCDE = PW
DD DD	LD BCDE,PX	BCDE = PX
DD ED	LD BCDE,PY	BCDE = PY
DD FD	LD BCDE,PZ	BCDE = PZ

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

The 32-bit register BCDE is loaded with `ps` (any of the 32-bit registers PW, PX, PY or PZ).

LD BCDE,(HL)

Opcode	Instruction	Operation

DD 1A	LD BCDE,(HL)	E = (HL)
		D = (HL + 1)
		C = (HL + 2)
		B = (HL + 3)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	14	n/a	n/a
Rabbit 5000	14	14	12

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•		•	

Description

The 32-bit register BCDE is loaded with the 4 bytes of data whose address starts at HL.

LD BCDE,(IX+*d*)
LD BCDE,(IY+*d*)

Opcode	Instruction	Operation

DD CE *d*	LD BCDE,(IX+*d*)	E = (IX + *d*)
		D = (IX + *d* + 1)
		C = (IX + *d* + 2)
		B = (IX + *d* + 3)

DD DE *d*	LD BCDE,(IY+*d*)	E = (IY + *d*)
		D = (IY + *d* + 1)
		C = (IY + *d* + 2)
		B = (IY + *d* + 3)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	15	n/a	n/a
Rabbit 5000	16	15	14

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•		•	

Description

Loads BCDE with the 4 bytes of data whose address starts at:

- the sum of IX and the 8-bit signed displacement *d*, or
- the sum of IY and the 8-bit signed displacement *d*.

LD BCDE,(*mn*)

Opcode	Instruction	Operation

93 *n m*	LD BCDE,(*mn*)	E = (*mn*)
		D = (*mn* + 1)
		C = (*mn* + 2)
		B = (*mn* + 3)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	15	n/a	n/a
Rabbit 5000	15	13	13

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•		•	

Description

Loads BCDE with the 4 bytes of data whose address starts at the 16-bit constant *mn*.

LD BCDE,(*ps*+*d*)

Opcode	Instruction	Operation
—	**LD BCDE,(*ps*+*d*)**	**E = (ps+*d*); D = (ps+*d*+1)** **C = (ps+*d*+2); B = (ps+*d*+3)**

DD 0E *d*	LD BCDE,(PW+*d*)	E = (PW+*d*); D = (PW+*d*+1)
		C = (PW+*d*+2); B = (PW+*d*+3)

DD 1E *d*	LD BCDE,(PX+*d*)	E = (PX+*d*); D = (PX+*d*+1)
		C = (PX+*d*+2); B = (PX+*d*+3)

DD 2E *d*	LD BCDE,(PY+*d*)	E = (PY+*d*); D = (PY+*d*+1)
		C = (PY+*d*+2); B = (PY+*d*+3)

DD 3E *d*	LD BCDE,(PZ+*d*)	E = (PZ+*d*); D = (PZ+*d*+1)
		C = (PZ+*d*+2); B = (PZ+*d*+3)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	15	n/a	n/a
Rabbit 5000	16	15	14

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads the 32-bit register BCDE with the data whose address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If *ps* is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 or 24 bits (depending on the memory that is used) being significant.

The address is computed as the sum of *ps* and the 8-bit signed displacement *d*.

`LD BCDE,(ps+HL)`

Opcode	Instruction	Operation
—	**LD BCDE,(*ps*+HL)**	**E = (*ps*+HL); D = (*ps*+HL+1) C = (ps+HL+2); B = (*ps*+HL+3)**
DD 0C *d*	LD BCDE,(PW+HL)	E = (PW+HL); D = (PW+HL+1) C = (PW+HL+2); B = (PW+HL+3)
DD 1C *d*	LD BCDE,(PX+HL)	E = (PX+HL); D = (PX+HL+1) C = (PX+HL+2); B = (PX+HL+3)
DD 2C *d*	LD BCDE,(PY+HL)	E = (PY+HL); D = (PY+HL+1) C = (PY+HL+2); B = (PY+HL+3)
DD 3C *d*	LD BCDE,(PZ+HL)	E = (PZ+HL); D = (PZ+HL+1) C = (PZ+HL+2); B = (PZ+HL+3)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	14	n/a	n/a
Rabbit 5000	15	15	13

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads the 32-bit register BCDE with the data whose address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If *ps* is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 or 24 bits (depending on the memory that is used) being significant.

The address is computed as the sum of *ps* and HL. HL is considered to be sign extended to 24 bits.

`LD BCDE,d`

Opcode	Instruction	Operation
A3 *d*	LD BCDE,*d*	BCDE = *d* (sign-extended to 32 bits)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
–	–	–	–		•			

Description

Loads the 32-bit register BCDE with *d*, the 8-bit constant sign extended to 32 bits.

LD BCDE,(SP+HL)

Opcode	Instruction	Operation

DD FE	LD BCDE,(SP+HL)	E = (SP + HL)
		D = (SP + HL + 1)
		C = (SP + HL + 2)
		B = (SP + HL + 3)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	14	n/a	n/a
Rabbit 5000	15	15	13

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads the 32-bit register BCDE with the data whose address is the sum of SP and HL.

LD BCDE,(SP+*n*)

Opcode	Instruction	Operation

DD EE *n*	LD BCDE,(SP+*n*)	E = (SP + *n*)
		D = (SP + *n* + 1)
		C = (SP + *n* + 2)
		B = (SP + *n* + 3)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	15	n/a	n/a
Rabbit 5000	16	15	14

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads the 32-bit register BCDE with the data whose address is the sum of SP and the 8-bit unsigned constant *n*.

LD BC,HL

Opcode	Instruction	Operation
91	LD BC,HL	BC = HL

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads BC with HL.

`LD dd',BC`
`LD dd',DE`

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
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 BC or DE.

`LD dd,mn`

Opcode	Instruction	Operation
——	**LD *dd,mn***	***dd* = *mn***
01 *n m*	LD BC,*mn*	BC = *mn*
11 *n m*	LD DE,*mn*	DE = *mn*
21 *n m*	LD HL,*mn*	HL = *mn*
31 *n m*	LD SP,*mn*	SP = *mn*

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	6	n/a	n/a
Rabbit 5000	6	4	4

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads *dd* (any of the 16-bit registers BC, DE, HL, or SP) with the 16-bit constant *mn*.

LD *dd*,(*mn*)

Opcode	Instruction	Operation
——	**LD *dd*,(*mn*)**	**$dd_{\texttt{low}}$ = (*mn*)** **$dd_{\texttt{high}}$ = (*mn* + 1)**

ED 4B *n* *m*	LD BC,(*mn*)	C = (*mn*)
		B = (*mn* + 1)

ED 5B *n* *m*	LD DE,(*mn*)	E = (*mn*)
		D = (*mn* + 1)

ED 6B *n* *m*	LD HL,(*mn*)[a]	L = (*mn*)
		H = (*mn* + 1)

ED 7B *n* *m*	LD SP,(*mn*)	$SP_{\texttt{low}}$ = (*mn*) $SP_{\texttt{high}}$ = (*mn* + 1)

a. A faster 3-byte version of LD HL,(mn) exists; this is the opcode that is generated by the assembler. See LD HL,(mn) for more information.

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	13	n/a	n/a
Rabbit 5000	13	11	9

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•		•	

Description

Loads *dd* (any of the 16-bit registers BC, DE, HL or SP) with the data whose address is the 16-bit constant *mn*.

LD DE,HL

Opcode	Instruction	Operation
B1	LD DE,HL	DE = HL

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads DE with HL.

LD EIR,A
LD IIR,A

Opcode	Instruction	Operation
ED 47	LD EIR,A	EIR = A
ED 4F	LD IIR,A	IIR = A

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

- `LD EIR,A:` Loads the External Interrupt Register, EIR, with 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 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,BC
LD HL,DE

Opcode	Instruction	Operation
81	LD HL,BC	HL = BC
A1	LD HL,DE	HL = DE

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
`S`	`Z`	`L/V`	`C`	`F`	`R`	`SP`	`S`	`D`
-	-	-	-		•			

Description

Loads HL with BC or DE.

LD HL,IX
LD HL,IY

Opcode	Instruction	Operation
DD 7C	LD HL,IX	HL = IX
FD 7C	LD HL,IY	HL = IY

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads HL with IX or IY.

`LD HL,(mn)`

Opcode	Instruction	Operation

2A *n m*	LD HL,(*mn*)	L = (*mn*)
		H = (*mn* + 1)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	11	n/a	n/a
Rabbit 5000	11	9	9

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
–	–	–	–		•		•	

Description

Loads HL with the data whose address is the 16-bit constant mn.

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

Opcode	Instruction	Operation

DD E4 *d*	LD HL,(HL+*d*)	L = (HL + *d*)
		H = (HL + *d* + 1)

FD E4 *d*	LD HL,(IY+*d*)	L = (IY + *d*)
		H = (IY + *d* + 1)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	11	n/a	n/a
Rabbit 5000	12	11	10

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
–	–	–	–		•		•	

Description

Loads HL with the data whose address is

- the sum of HL and the 8-bit signed displacement *d*, or
- the sum of IY and the 8-bit signed displacement *d*.

`LD HL,(IX+d)`

Opcode	Instruction	Operation

E4 *d*	LD HL,(IX+*d*)	L = (IX + *d*)
		H = (IX + *d* + 1)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	9	n/a	n/a
Rabbit 5000	10	10	9

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
–	–	–	–		•		•	

Description

Loads HL with the data whose address is the sum of IX and the 8-bit signed displacement *d*.

`LD HL,(`*`ps`*`+BC)`

Opcode	Instruction	Operation
—	**LD HL,(*ps*+BC)**	**L = (*ps* + BC)** **H = (*ps* + BC + 1)**

ED 06	LD HL,(PW+BC)	L = (PW + BC)
		H = (PW + BC + 1)

ED 16	LD HL,(PX+BC)	L = (PX + BC)
		H = (PX + BC + 1)

ED 26	LD HL,(PY+BC)	L = (PY + BC)
		H = (PY + BC + 1)

ED 36	LD HL,(PZ+BC)	L = (PZ + BC)
		H = (PZ + BC + 1)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	10	n/a	n/a
Rabbit 5000	11	11	9

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads HL with the data whose address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If *ps* is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

The address is computed as the sum of *ps* and BC. BC is considered to be sign extended to 24 bits.

LD HL,(*ps*+*d*)

Opcode	Instruction	Operation
—	**LD HL,(*ps*+*d*)**	**L = (*ps* + *d*)** **H = (*ps* + *d* + 1)**

85 *d*	LD HL,(PW+*d*)	L = (PW + *d*)
		H = (PW + *d* + 1)

95 *d*	LD HL,(PX+*d*)	L = (PX + *d*)
		H = (PX + *d* + 1)

A5 *d*	LD HL,(PY+*d*)	L = (PY + *d*)
		H = (PY + *d* + 1)

B5 *d*	LD HL,(PZ+*d*)	L = (PZ + *d*)
		H = (PZ + *d* + 1)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	9	n/a	n/a
Rabbit 5000	10	10	9

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads HL with the data whose address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If *ps* is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

The address is computed as the sum of *ps* and the 8-bit signed displacement *d*.

LD HL,(SP+HL)

Opcode	Instruction	Operation

ED FE	LD HL,(SP+HL)	L = (SP + HL)
		H = (SP + HL + 1)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	10	n/a	n/a
Rabbit 5000	11	11	9

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads HL with the data whose address is the sum of SP and HL.

LD HL,LXPC

Opcode	Instruction	Operation
9F	LD HL,LXPC	HL = LXPC

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads HL with the extended 12-bit XPC (LXPC). This is a chained-atomic instruction, meaning that an interrupt cannot take place between this instruction and the instruction following it.

Load 4000, 5000

LD HTR,A

Opcode	Instruction	Operation
ED 40	LD HTR,A	HTR = A

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads HTR with A.

LD HL,(SP+*n*)

Opcode	Instruction	Operation

C4 *n*	LD HL,(SP+*n*)	L = (SP + *n*)
		H = (SP + *n* + 1)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	9	n/a	n/a
Rabbit 5000	10	10	9

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads HL with the data whose address is the sum of SP and the 8-bit unsigned constant *n*.

LD IX,HL
LD IY,HL

Opcode	Instruction	Operation
DD 7D	LD IX,HL	IX = HL
FD 7D	LD IY,HL	IY = HL

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads IX or IY with HL.

`LD IX,`*`mn`*
`LD IY,`*`mn`*

Opcode	Instruction	Operation
DD 21 *n* *m*	LD IX,*mn*	IX = *mn*
FD 21 *n* *m*	LD IY,*mn*	IY = *mn*

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	8	n/a	n/a
Rabbit 5000	8	6	4

Flags				ALTD			IOI/IOE	
`S`	`Z`	`L/V`	`C`	`F`	`R`	`SP`	`S`	`D`
-	-	-	-					

Description

Loads IX or IY with the 16-bit constant *mn*.

LD IX,(*mn*)

Opcode	Instruction	Operation
DD 2A *n m*	LD IX,(*mn*)	IX_{low} = (*mn*) IX_{high} = (*mn* + 1)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	13	n/a	n/a
Rabbit 5000	13	11	9

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-				•	

Description

Loads IX with the data whose address is the 16-bit constant *mn*.

LD IX,(SP+*n*)

Opcode	Instruction	Operation
DD C4 *n*	LD IX,(SP+*n*)	IX_{low} = (SP + *n*) IX_{high} = (SP + *n* + 1)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	11	n/a	n/a
Rabbit 5000	12	11	10

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads IX with the data whose address is the sum of SP and the 8-bit unsigned constant *n*.

`LD IY,(`*`mn`*`)`

Opcode	Instruction	Operation
FD 2A *n* *m*	LD IY,(*mn*)	IY_{low} = (*mn*) IY_{high} = (*mn* + 1)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	13	n/a	n/a
Rabbit 5000	13	11	9

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-				•	

Description

Loads IY with the data whose address is the 16-bit constant *mn*.

LD IY,(SP+*n*)

Opcode	Instruction	Operation
FD C4 *n*	LD IY,(SP+*n*)	IY_{low} = (SP + *n*) IY_{high} = (SP + *n* + 1)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	11	n/a	n/a
Rabbit 5000	12	11	10

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads IY with the data whose address is the sum of SP and the 8-bit unsigned constant *n*.

LD JKHL,*d*

Opcode	Instruction	Operation
A4 *d*	LD JKHL,*d*	JKHL = *d* (sign-extended to 32 bits)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads JKHL with the 8-bit value *d*, sign-extended to 32 bits.

LD JKHL,*ps*

Opcode	Instruction	Operation
—	**LD JKHL,*ps***	**JKHL = *ps***
FD CD	LD JKHL,PW	JKHL = PW
FD DD	LD JKHL,PX	JKHL = PX
FD ED	LD JKHL,PY	JKHL = PY
FD FD	LD JKHL,PZ	JKHL = PZ

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads JKHL with *ps* (any of the 32-bit registers PW, PX, PY or PZ).

LD JKHL,(HL)

Opcode	Instruction	Operation
FD 1A	LD JKHL,(HL)	JKHL= (HL)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	14	n/a	n/a
Rabbit 5000	14	14	12

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
–	–	–	–		•		•	

Description

Loads JKHL with the data whose address is in HL.

LD JKHL,(SP+HL)

Opcode	Instruction	Operation
FD FE	LD JKHL,(SP+HL)	JKHL= (SP + HL)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	14	n/a	n/a
Rabbit 5000	15	15	13

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads JKHL with the data whose address is the sum of SP and HL.

LD JKHL,(IX+*d*)
LD JKHL,(IY+*d*)

Opcode	Instruction	Operation
FD CE *d*	LD JKHL,(IX+*d*)	JKHL= (IX + *d*)
FD DE *d*	LD JKHL,(IY+*d*)	JKHL= (IY + *d*)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	15	n/a	n/a
Rabbit 5000	16	15	14

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
–	–	–	–		•		•	

Description

Loads JKHL with the data whose address is

• the sum of IX and the 8-bit signed displacement *d*, or

• the sum of IY and the 8-bit signed displacement *d*

LD JKHL,(*mn*)

Opcode	Instruction	Operation
94 *n* *m*	LD JKHL,(*mn*)	JKHL= (*mn*)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	15	n/a	n/a
Rabbit 5000	15	13	13

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•		•	

Description

Loads JKHL with the data whose address is the 16-bit constant *mn*.

LD JKHL, (*ps*+*d*)

Opcode	Instruction	Operation
—	**LD JKHL,(*ps*+*d*)**	**JKHL= (*ps*+*d*)**
FD 0E *d*	LD JKHL,(PW+*d*)	JKHL= (PW+*d*)
FD 1E *d*	LD JKHL,(PX+*d*)	JKHL= (PX+*d*)
FD 2E *d*	LD JKHL,(PY+*d*)	JKHL= (PY+*d*)
FD 3E *d*	LD JKHL,(PZ+*d*)	JKHL= (PZ+*d*)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	15	n/a	n/a
Rabbit 5000	16	15	14

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads JKHL with the data whose address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If *ps* is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a "long logical" address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

The address is computed as the sum of *ps* and the 8-bit signed displacement *d*.

LD JKHL,(*ps*+HL)

Opcode	Instruction	Operation
—	**LD JKHL,(*ps*+HL)**	**JKHL=(*ps*+HL)**
FD 0C *d*	LD JKHL,(PW+HL)	JKHL= (PW+HL)
FD 1C *d*	LD JKHL,(PX+HL)	JKHL= (PX+HL)
FD 2C *d*	LD JKHL,(PY+HL)	JKHL= (PY+HL)
FD 3C *d*	LD JKHL,(PZ+HL)	JKHL= (PZ+HL)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	14	n/a	n/a
Rabbit 5000	15	15	13

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads JKHL with the data whose address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If *ps* is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

The address is computed as the sum of *ps* and HL. HL is considered sign extended to 24 bits.

LD JKHL,(SP+*n*)

Opcode	Instruction	Operation

FD EE *n*	LD JKHL,(SP+*n*)	L = (SP + *n*)
		H = (SP + *n* + 1)
		K = (SP + *n* + 2)
		J = (SP + *n* + 3)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	15	n/a	n/a
Rabbit 5000	16	15	14

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
–	–	–	–		•			

Description

Loads JKHL with the data whose address is the sum of SP and the 8-bit unsigned constant *n*.

LD JK,*mn*

Opcode	Instruction	Operation
A9 *n m*	LD JK,*mn*	JK = *mn*

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	6	n/a	n/a
Rabbit 5000	6	4	4

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads JK with the 16-bit constant *mn*.

LD JK,(*mn*)

Opcode	Instruction	Operation

99 *n* *m*	LD JK,(*mn*)	J = (*mn*)
		K = (*mn* + 1)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	11	n/a	n/a
Rabbit 5000	11	9	9

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
–	–	–	–		•		•	

Description

Loads JK with the data whose address is *mn*.

LD *pd*,BCDE

Opcode	Instruction	Operation
—	**LD *pd*,BCDE**	**pd = BCDE**
DD 8D	LD PW,BCDE	PW = BCDE
DD 9D	LD PX,BCDE	PX = BCDE
DD AD	LD PY,BCDE	PY = BCDE
DD BD	LD PZ,BCDE	PZ = BCDE

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads *pd* (any of the 32-bit registers PW, PX, PY or PZ) with BCDE.

LD *pd*,JKHL

Opcode	Instruction	Operation
—	**LD *pd*,JKHL**	**pd = JKHL**
FD 8D	LD PW,JKHL	PW = JKHL
FD 9D	LD PX,JKHL	PX = JKHL
FD AD	LD PY,JKHL	PY = JKHL
FD BD	LD PZ,JKHL	PZ = JKHL

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads *pd* (any of the 32-bit registers PW, PX, PY or PZ) with JKHL.

LD *pd,klmn*

Opcode	Instruction	Operation
—	**LD *pd,klmn***	**pd = klmn**
ED 0C *n m l k*	LD PW,*klmn*	PW_0=*n*; PW_1=*m*; PW_2=*l*; PW_3=*k*
ED 1C *n m l k*	LD PX,*klmn*	PX_0=*n*; PX_1=*m*; PX_2=*l*; PX_3=*k*
ED 2C *n m l k*	LD PY,*klmn*	PY_0=*n*; PY_1=*m*; PY_2=*l*; PY_3=*k*
ED 3C *n m l k*	LD PZ,*klmn*	PZ_0=*n*; PZ_1=*m*; PZ_2=*l*; PZ_3=*k*

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	12	n/a	n/a
Rabbit 5000	12	8	6

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads *pd* (any of the 32-bit registers PW, PX, PY or PZ) with the 32-bit constant *klmn*.

LD *pd,ps*

Opcode	Instruction	Operation
—	**LD *pd,ps***	**pd = *ps***
6D 07	LD PW,PW	PW = PW
6D 17	LD PW,PX	PW = PX
6D 27	LD PW,PY	PW = PY
6D 37	LD PW,PZ	PW = PZ
6D 47	LD PX,PW	PX = PW
6D 57	LD PX,PX	PX = PX
6D 67	LD PX,PY	PX = PY
6D 77	LD PX,PZ	PX = PZ
6D 87	LD PY,PW	PY = PW
6D 97	LD PY,PX	PY = PX
6D A7	LD PY,PY	PY = PY
6D B7	LD PY,PZ	PY = PZ
6D C7	LD PZ,PW	PZ = PW
6D D7	LD PZ,PX	PZ = PX
6D E7	LD PZ,PY	PZ = PY
6D F7	LD PZ,PZ	PZ = PZ

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads *pd* (any of the 32-bit registers PW, PX, PY or PZ) with *ps* (any of PW, PX, PY or PZ).

LD *pd*,*ps*+*d*

Opcode	Instruction	Operation
—	**LD *pd*,*ps*+*d***	***pd* = *ps* +*d***
6D 0C *d*	LD PW,PW+*d*	PW = PW + *d*
6D 1C *d*	LD PW,PX+*d*	PW = PX + *d*
6D 2C *d*	LD PW,PY+*d*	PW = PY + *d*
6D 3C *d*	LD PW,PZ+*d*	PW = PZ + *d*
6D 4C *d*	LD PX,PW+*d*	PX = PW + *d*
6D 5C *d*	LD PX,PX+*d*	PX = PX + *d*
6D 6C *d*	LD PX,PY+*d*	PX = PY + *d*
6D 7C *d*	LD PX,PZ+*d*	PX = PZ + *d*
6D 8C *d*	LD PY,PW+*d*	PY = PW + *d*
6D 9C *d*	LD PY,PX+*d*	PY = PX + *d*
6D AC *d*	LD PY,PY+*d*	PY = PY + *d*
6D BC *d*	LD PY,PZ+*d*	PY = PZ + *d*
6D CC *d*	LD PZ,PW+*d*	PZ = PW + *d*
6D DC *d*	LD PZ,PX+*d*	PZ = PX + *d*
6D EC *d*	LD PZ,PY+*d*	PZ = PY + *d*
6D FC *d*	LD PZ,PZ+*d*	PZ = PZ + *d*

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	6	n/a	n/a
Rabbit 5000	6	4	4

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads *pd* (any of the 32-bit registers PW, PX, PY or PZ) with the sum of *ps* (any of PW, PX, PY or PZ) and the 8-bit displacement *d*. These instructions cannot be used for general 32-bit arithmetic because the addition depends on the upper two bytes of *ps*. If the upper two bytes are all 1's, then it is 16-bit addition. The following example illustrates this point:

```
ld PW,0xFFFFFFFF
ld PW,PW+1              ;yields PW=0xFFFF0000

ld PW,0x7FFFFFFF
ld PW,PW+1              ;yields PW=0x80000000
```

LD *pd*,*ps*+DE

Opcode	Instruction	Operation
—	**LD *pd*,*ps*+DE**	***pd* = *ps* + DE**
6D 06	LD PW,PW+DE	PW = PW + DE
6D 16	LD PW,PX+DE	PW = PX + DE
6D 26	LD PW,PY+DE	PW = PY + DE
6D 36	LD PW,PZ+DE	PW = PZ + DE
6D 46	LD PX,PW+DE	PX = PW + DE
6D 56	LD PX,PX+DE	PX = PX + DE
6D 66	LD PX,PY+DE	PX = PY + DE
6D 76	LD PX,PZ+DE	PX = PZ + DE
6D 86	LD PY,PW+DE	PY = PW + DE
6D 96	LD PY,PX+DE	PY = PX + DE
6D A6	LD PY,PY+DE	PY = PY + DE
6D B6	LD PY,PZ+DE	PY = PZ + DE
6D C6	LD PZ,PW+DE	PZ = PW + DE
6D D6	LD PZ,PX+DE	PZ = PX + DE
6D E6	LD PZ,PY+DE	PZ = PY + DE
6D F6	LD PZ,PZ+DE	PZ = PZ + DE

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads *pd* (any of the 32-bit registers PW, PX, PY or PZ) with the sum of *ps* (any of PW, PX, PY or PZ) and DE. These instructions cannot be used for general 32-bit arithmetic because the addition depends on the upper two bytes of *ps*. If the upper two bytes are all ones, then it is 16-bit addition. The following example illustrates this point:

```
ld PW,0xFFFFFFFF
ld PW,PW+DE             ;yields PW=0xFFFF0000 if DE=1

ld PW,0x7FFFFFFF
ld PW,PW+DE             ;yields PW=0x80000000 if DE=1
```

LD *pd*,*ps*+HL

Opcode	Instruction	Operation
—	**LD *pd*,*ps*+HL**	***pd* = *ps* + HL**
6D 0E	LD PW,PW+HL	PW = PW + HL
6D 1E	LD PW,PX+HL	PW = PX + HL
6D 2E	LD PW,PY+HL	PW = PY + HL
6D 3E	LD PW,PZ+HL	PW = PZ + HL
6D 4E	LD PX,PW+HL	PX = PW + HL
6D 5E	LD PX,PX+HL	PX = PX + HL
6D 6E	LD PX,PY+HL	PX = PY + HL
6D 7E	LD PX,PZ+HL	PX = PZ + HL
6D 8E	LD PY,PW+HL	PY = PW + HL
6D 9E	LD PY,PX+HL	PY = PX + HL
6D AE	LD PY,PY+HL	PY = PY + HL
6D BE	LD PY,PZ+HL	PY = PZ + HL
6D CE	LD PZ,PW+HL	PZ = PW + HL
6D DE	LD PZ,PX+HL	PZ = PX + HL
6D EE	LD PZ,PY+HL	PZ = PY + HL
6D FE	LD PZ,PZ+HL	PZ = PZ + HL

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads *pd* (any of the 32-bit registers PW, PX, PY or PZ) with the sum of *ps* (any of PW, PX, PY or PZ) and HL. These instructions cannot be used for general 32-bit arithmetic because the addition depends on the upper two bytes of *ps*. If the upper two bytes are all ones, then it is 16-bit addition. The following example illustrates this point:

```
ld PW,0xFFFFFFFF
ld PW,PW+HL             ;yields PW=0xFFFF0000 if HL=1

ld PW,0x7FFFFFFF
ld PW,PW+HL             ;yields PW=0x80000000 if HL=1
```

LD *pd*,*ps*+IX

Opcode	Instruction	Operation
—	**LD *pd*,*ps*+IX**	**pd = *ps* + IX**
6D 04	LD PW,PW+IX	PW = PW + IX
6D 14	LD PW,PX+IX	PW = PX + IX
6D 24	LD PW,PY+IX	PW = PY + IX
6D 34	LD PW,PZ+IX	PW = PZ + IX
6D 44	LD PX,PW+IX	PX = PW + IX
6D 54	LD PX,PX+IX	PX = PX + IX
6D 64	LD PX,PY+IX	PX = PY + IX
6D 74	LD PX,PZ+IX	PX = PZ + IX
6D 84	LD PY,PW+IX	PY = PW + IX
6D 94	LD PY,PX+IX	PY = PX + IX
6D A4	LD PY,PY+IX	PY = PY + IX
6D B4	LD PY,PZ+IX	PY = PZ + IX
6D C4	LD PZ,PW+IX	PZ = PW + IX
6D D4	LD PZ,PX+IX	PZ = PX + IX
6D E4	LD PZ,PY+IX	PZ = PY + IX
6D F4	LD PZ,PZ+IX	PZ = PZ + IX

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads *pd* (any of the 32-bit registers PW, PX, PY or PZ) with the sum of *ps* (any of PW, PX, PY or PZ) and IX. These instructions cannot be used for general 32-bit arithmetic because the addition depends on the upper two bytes of *ps*. If the upper two bytes are all ones, then it is 16-bit addition. The following example illustrates this point:

```
ld PW,0xFFFFFFFF
ld PW,PW+IX            ;yields PW=0xFFFF0000 if IX=1

ld PW,0x7FFFFFFF
ld PW,PW+IX            ;yields PW=0x80000000 if IX=1
```

LD *pd*,*ps*+IY

Opcode	Instruction	Operation
—	**LD *pd*,*ps*+IY**	**pd = *ps* + IY**
6D 05	LD PW,PW+IY	PW = PW + IY
6D 15	LD PW,PX+IY	PW = PX + IY
6D 25	LD PW,PY+IY	PW = PY + IY
6D 35	LD PW,PZ+IY	PW = PZ + IY
6D 45	LD PX,PW+IY	PX = PW + IY
6D 55	LD PX,PX+IY	PX = PX + IY
6D 65	LD PX,PY+IY	PX = PY + IY
6D 75	LD PX,PZ+IY	PX = PZ + IY
6D 85	LD PY,PW+IY	PY = PW + IY
6D 95	LD PY,PX+IY	PY = PX + IY
6D A5	LD PY,PY+IY	PY = PY + IY
6D B5	LD PY,PZ+IY	PY = PZ + IY
6D C5	LD PZ,PW+IY	PZ = PW + IX
6D D5	LD PZ,PX+IY	PZ = PX + IX
6D E5	LD PZ,PY+IY	PZ = PY + IX
6D F5	LD PZ,PZ+IY	PZ = PZ + IX

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads *pd* (any of the 32-bit registers PW, PX, PY or PZ) with the sum of *ps* (any of PW, PX, PY or PZ) and IY. These instructions cannot be used for general 32-bit arithmetic because the addition depends on the upper two bytes of *ps*. If the upper two bytes are all ones, then it is 16-bit addition. The following example illustrates this point:

```
ld PW,0xFFFFFFFF
ld PW,PW+IY            ;yields PW=0xFFFF0000 if IY=1

ld PW,0x7FFFFFFF
ld PW,PW+IY            ;yields PW=0x80000000 if IY=1
```

`LD pd,(HTR+HL)`

Opcode	Instruction	Operation
—	**LD *pd*,(HTR+HL)**	**pd_0 = (HTR + HL)** **pd_1 = (HTR + HL + 1)** **pd_2 = (HTR + HL + 2)** **pd_3 = (HTR + HL + 3)**
ED 01	LD PW,(HTR+HL)	PW_0 = (HTR + HL) PW_1 = (HTR + HL + 1) PW_2 = (HTR + HL + 2) PW_3 = (HTR + HL + 3)
ED 11	LD PX,(HTR+HL)	PX_0 = (HTR + HL) PX_1 = (HTR + HL + 1) PX_2 = (HTR + HL + 2) PX_3 = (HTR + HL + 3)
ED 21	LD PY,(HTR+HL)	PY_0 = (HTR + HL) PY_1 = (HTR + HL + 1) PY_2 = (HTR + HL + 2) PY_3 = (HTR + HL + 3)
ED 31	LD PZ,(HTR+HL)	PZ_0 = (HTR + HL) PZ_1 = (HTR + HL + 1) PZ_2 = (HTR + HL + 2) PZ_3 = (HTR + HL + 3)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	14	n/a	n/a
Rabbit 5000	15	15	13

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads *pd* (any of the 32-bit registers PW, PX, PY or PZ) with the data whose address is the sum of HTR and HL.

LD *pd*, (*ps*+*d*)

Opcode	Instruction	Operation
—	**LD *pd*,(*ps*+*d*)**	**pd_0=(*ps*+*d*); pd_1=(*ps*+*d*+1) pd_2=(*ps*+*d*+2); pd_3=(*ps*+*d*+3)**
6D 08 *d*	LD PW,(PW+*d*)	PW_0=(*ps*+*d*) PW_1=(*ps*+*d*+1) PW_2=(*ps*+*d*+2) PW_3=(*ps*+*d*+3)
6D 18 *d*	LD PW,(PX+*d*)	
6D 28 *d*	LD PW,(PY+*d*)	
6D 38 *d*	LD PW,(PZ+*d*)	
6D 48 *d*	LD PX,(PW+*d*)	PX_0=(*ps*+*d*) PX_1=(*ps*+*d*+1) PX_2=(*ps*+*d*+2) PX_3=(*ps*+*d*+3)
6D 58 *d*	LD PX,(PX+*d*)	
6D 68 *d*	LD PX,(PY+*d*)	
6D 78 *d*	LD PX,(PZ+*d*)	
6D 88 *d*	LD PY,(PW+*d*)	PY_0=(*ps*+*d*) PY_1=(*ps*+*d*+1) PY_2=(*ps*+*d*+2) PY_3=(*ps*+*d*+3)
6D 98 *d*	LD PY,(PX+*d*)	
6D A8 *d*	LD PY,(PY+*d*)	
6D B8 *d*	LD PY,(PZ+*d*)	
6D C8 *d*	LD PZ,(PW+*d*)	PZ_0=(*ps*+*d*) PZ_1=(*ps*+*d*+1) PZ_2=(*ps*+*d*+2) PZ_3=(*ps*+*d*+3)
6D D8 *d*	LD PZ,(PX+*d*)	
6D E8 *d*	LD PZ,(PY+*d*)	
6D F8 *d*	LD PZ,(PZ+*d*)	

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	15	n/a	n/a
Rabbit 5000	16	15	14

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads *pd* (any of the 32-bit registers PW, PX, PY or PZ) with the data whose address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If *`ps`* is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

The address is computed as the sum of *ps* and the 8-bit displacement *d*.

LD *pd*, (*ps*+HL)

Opcode	Instruction	Operation
—	**LD *pd*,(*ps*+HL)**	**pd_0=(*ps*+HL); pd_1=(*ps*+HL+1) pd_2=(*ps*+HL+2); pd_3=(*ps*+HL+3)**
6D 0A	LD PW,(PW+HL)	PW_0=(*ps*+HL) PW_1=(*ps*+HL+1) PW_2=(*ps*+HL+2) PW_3=(*ps*+HL+3)
6D 1A	LD PW,(PX+HL)	
6D 2A	LD PW,(PY+HL)	
6D 3A	LD PW,(PZ+HL)	
6D 4A	LD PX,(PW+HL)	PX_0=(*ps*+HL) PX_1=(*ps*+HL+1) PX_2=(*ps*+HL+2) PX_3=(*ps*+HL+3)
6D 5A	LD PX,(PX+HL)	
6D 6A	LD PX,(PY+HL)	
6D 7A	LD PX,(PZ+HL)	
6D 8A	LD PY,(PW+HL)	PY_0=(*ps*+HL) PY_1=(*ps*+HL+1) PY_2=(*ps*+HL+2) PY_3=(*ps*+HL+3)
6D 9A	LD PY,(PX+HL)	
6D AA	LD PY,(PY+HL)	
6D BA	LD PY,(PZ+HL)	
6D CA	LD PZ,(PW+HL)	PZ_0=(*ps*+HL) PZ_1=(*ps*+HL+1) PZ_2=(*ps*+HL+2) PZ_3=(*ps*+HL+3)
6D DA	LD PZ,(PX+HL)	
6D EA	LD PZ,(PY+HL)	
6D FA	LD PZ,(PZ+HL)	

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	14	n/a	n/a
Rabbit 5000	15	15	13

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads *pd* (any of the 32-bit registers PW, PX, PY or PZ) with the data whose address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If `ps` is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

The address is computed as the sum of *ps* and HL. HL is considered sign extended to 24 bits.

LD *pd*, (SP+*n*)

Opcode	Instruction	Operation
—	**LD *pd*,(SP+*n*)**	**pd_0=(SP+*n*)** **pd_1=(SP+*n*+1)** **pd_2=(SP+*n*+2)** **pd_3=(SP+*n*+3)**
ED 04 *n*	LD PW,(SP+*n*)	PW_0=(SP+*n*) PW_1=(SP+*n*+1) PW_2=(SP+*n*+2) PW_3=(SP+*n*+3)
ED 14 *n*	LD PX,(SP+*n*)	PX_0=(SP+*n*) PX_1=(SP+*n*+1) PX_2=(SP+*n*+2) PX_3=(SP+*n*+3)
ED 24 *n*	LD PY,(SP+*n*)	PY_0=(SP+*n*) PY_1=(SP+*n*+1) PY_2=(SP+*n*+2) PY_3=(SP+*n*+3)
ED 34 *n*	LD PZ,(SP+*n*)	PZ_0=(SP+*n*) PZ_1=(SP+*n*+1) PZ_2=(SP+*n*+2) PZ_3=(SP+*n*+3)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	15	n/a	n/a
Rabbit 5000	16	15	14

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads *pd* (any of the 32-bit registers PW, PX, PY or PZ) with the 32 bits of data whose address is the sum of SP and the 8-bit unsigned constant *n*.

LD *rr*,(*ps*+*d*)

Opcode	Instruction	Operation
—	**LD *rr*,(*ps*+*d*)**	**rr_{low} = (ps+d); rr_{high} =(ps+d+1)**
6D 00 *d*	LD BC,(PW+*d*)	C = (PW+*d*); B = (PW+*d*+1)
6D 10 *d*	LD BC,(PX+*d*)	C = (PX+*d*); B = (PX+*d*+1)
6D 20 *d*	LD BC,(PY+*d*)	C = (PY+*d*); B = (PY+*d*+1)
6D 30 *d*	LD BC,(PZ+*d*)	C = (PZ+*d*); B = (PZ+*d*+1)
6D 40 *d*	LD DE,(PW+*d*)	E = (PW+*d*); D = (PW+*d*+1)
6D 50 *d*	LD DE,(PX+*d*)	E = (PX+*d*); D = (PX+*d*+1)
6D 60 *d*	LD DE,(PY+*d*)	E = (PY+*d*); D = (PY+*d*+1)
6D 70 *d*	LD DE,(PZ+*d*)	E = (PZ+*d*); D = (PZ+*d*+1)
6D 80 *d*	LD IX,(PW+*d*)	IX_{low}=(PW+*d*); IX_{high}=(PW+*d*+1)
6D 90 *d*	LD IX,(PX+*d*)	IX_{low}=(PX+*d*); IX_{high}=(PX+*d*+1)
6D A0 *d*	LD IX,(PY+*d*)	IX_{low}=(PY+*d*); IX_{high}=(PY+*d*+1)
6D B0 *d*	LD IX,(PZ+*d*)	IX_{low}=(PZ+*d*); IX_{high}=(PZ+*d*+1)
6D C0 *d*	LD IY,(PW+*d*)	IY_{low}=(PW+*d*); IY_{high}=(PW+*d*+1)
6D D0 *d*	LD IY,(PX+*d*)	IY_{low}=(PX+*d*); IY_{high}=(PX+*d*+1)
6D E0 *d*	LD IY,(PY+*d*)	IY_{low}=(PY+*d*); IY_{high}=(PY+*d*+1)
6D F0 *d*	LD IY,(PZ+*d*)	IY_{low}=(PZ+*d*); IY_{high}=(PZ+*d*+1)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	11	n/a	n/a
Rabbit 5000	12	11	10

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads *rr* (any of the 16-bit registers BC, DE, IX or IY) with the data whose address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation. If `ps` is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

The address is computed as the sum of *ps* (one of the 32-bit registers PW, PX, PY or PZ) and the 8-bit signed displacement *d.*

The instructions “LD IX,(ps+d)” and “LD IY,(ps+d)” are not affected by ALTD.

LD *rr*,(*ps*+HL)

Opcode	Instruction	Operation
—	**LD *rr*,(*ps*+HL)**	**rr_{low}= (ps+d); rr_{high} =(ps+d+1)**
6D 02	LD BC,(PW+HL)	C = (PW+HL); B = (PW+HL+1)
6D 12	LD BC,(PX+HL)	C = (PX+HL); B = (PX+HL+1)
6D 22	LD BC,(PY+HL)	C = (PY+HL); B = (PY+HL+1)
6D 32	LD BC,(PZ+HL)	C = (PZ+HL); B = (PZ+HL+1)
6D 42	LD DE,(PW+HL)	E = (PW+HL); D = (PW+HL+1)
6D 52	LD DE,(PX+HL)	E = (PX+HL); D = (PX+HL+1)
6D 62	LD DE,(PY+HL)	E = (PY+HL); D = (PY+HL+1)
6D 72	LD DE,(PZ+HL)	E = (PZ+HL); D = (PZ+HL+1)
6D 82	LD IX,(PW+HL)	IX_{low}=(PW+HL); IX_{high}=(PW+HL+1)
6D 92	LD IX,(PX+HL)	IX_{low}=(PX+HL); IX_{high}=(PX+HL+1)
6D A2	LD IX,(PY+HL)	IX_{low}=(PY+HL); IX_{high}=(PY+HL+1)
6D B2	LD IX,(PZ+HL)	IX_{low}=(PZ+HL); IX_{high}=(PZ+HL+1)
6D C2	LD IY,(PW+HL)	IY_{low}=(PW+HL); IY_{high}=(PW+HL+1)
6D D2	LD IY,(PX+HL)	IY_{low}=(PX+HL); IY_{high}=(PX+HL+1)
6D E2	LD IY,(PY+HL)	IY_{low}=(PY+HL); IY_{high}=(PY+HL+1)
6D F2	LD IY,(PZ+HL)	IY_{low}=(PZ+HL); IY_{high}=(PZ+HL+1)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	10	n/a	n/a
Rabbit 5000	11	11	9

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads *rr* (one of the 16-bit registers BC, DE, IX or IY) with the data whose address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If `ps` is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a "long logical" address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

The address is computed as the sum of *ps* (one of the 32-bit registers PW, PX, PY or PZ) and HL.

The instructions "LD IX,(ps+d)" and "LD IY,(ps+d)" are not affected by ALTD.

`LD r,g`

Opcode								Instruction	Operation
r,g	A	B	C	D	E	H	L	LD *r,g*	*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		

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000	2	n/a	n/a

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads *r* (any of the registers A, B, C, D, E, H, or L) with *g* (any of the registers A, B, C, D, E, H, or L).

LD *r*,*g*

Opcode								Instruction	Operation
r,g	A	B	C	D	E	H	L	LD *r*,*g*	*r* = *g*
A	7F 7F	7F 78	7F 79	7F 7A	7F 7B	7F 7C	7F 7D		
B	7F 47	7F 40	7F 41	7F 42	7F 43	7F 44	7F 45		
C	7F 4F	7F 48	7F 49	7F 4A	7F 4B	7F 4C	7F 4D		
D	7F 57	7F 50	7F 51	7F 52	7F 53	7F 54	7F 55		
E	7F 5F	7F 58	7F 59	7F 5A	5B[a]	7F 5C	7F 5D		
H	7F 67	7F 60	7F 61	7F 62	7F 63	7F 64	7F 65		
L	7F 6F	7F 68	7F 69	7F 6A	7F 6B	7F 6C	7F 6D		

a. This is actually the IDET instruction, unless preceded by the ALTD prefix, in which case the instruction is “LD E’,E”

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads *r* (any of the registers A, B, C, D, E, H, or L) with *g* (any of the registers A, B, C, D, E, H, or L).

LD *r*,*n*

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	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*,(HL)

Opcode	Instruction	Operation
——	**LD *r*,(HL)**	***r* = (HL)**
7E	LD A,(HL)	A = (HL)
46	LD B,(HL)	B = (HL)
4E	LD C,(HL)	C = (HL)
56	LD D,(HL)	D = (HL)
5E	LD E,(HL)	E = (HL)
66	LD H,(HL)	H = (HL)
6E	LD L,(HL)	L = (HL)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	5	n/a	n/a
Rabbit 5000	5	5	5

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	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.

`LD r,(IX+d)`
`LD r,(IY+d)`

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	9	n/a	n/a
Rabbit 5000	10	9	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•		•	

Description

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

- the sum of IX and a the 8-bit signed displacement *d*, or
- the sum of IY and *d*

LD SP,HL

Opcode	Instruction	Operation
F9	LD SP,HL	SP = HL

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads SP with HL.

These are chained-atomic instructions, meaning that an interrupt cannot take place between one of these instructions and the instruction following it.

LD SP,IX
LD SP,IY

Opcode	Instruction	Operation
DD F9	LD SP,IX	SP = IX
FD F9	LD SP,IY	SP = IY

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads SP with IX or IY.

These are chained-atomic instructions, meaning that an interrupt cannot take place between one of these instructions and the instruction following it.

LD XPC,A

Opcode	Instruction	Operation
ED 67	LD XPC,A	XPC = A

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads XPC with A.

This is a chained-atomic instruction, meaning that an interrupt cannot take place between this instruction and the instruction following it.

LD LXPC,HL

Opcode	Instruction	Operation
97	LD LXPC,HL	$LXPC_{low}$ = L $LXPC_{high}$ = H

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the 12-bit LXPC with HL. The most significant 4 bits of HL are ignored.

This is a chained-atomic instruction, meaning that an interrupt cannot take place between this instruction and the instruction following it.

LD (BC),A
LD (DE),A

Opcode	Instruction	Operation
02	LD (BC),A	(BC) = A
12	LD (DE),A	(DE) = A

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	7	n/a	n/a
Rabbit 5000	7	7	7

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
–	–	–	–					•

Description

Loads the memory location whose address is BC or DE with A.

`LD (HL),n`

Opcode	Instruction	Operation
36 *n*	LD (HL),*n*	(HL) = *n*

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	7	n/a	n/a
Rabbit 5000	7	7	6

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					•

Description

Loads the memory location whose address is HL with the 8-bit constant *n*.

LD (HL),*r*

Opcode	Instruction	Operation
——	**LD (HL),*r***	**(HL) = *r***
77	LD (HL),A	(HL) = A
70	LD (HL),B	(HL) = B
71	LD (HL),C	(HL) = C
72	LD (HL),D	(HL) = D
73	LD (HL),E	(HL) = E
74	LD (HL),H	(HL) = H
75	LD (HL),L	(HL) = L

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	6	n/a	n/a
Rabbit 5000	6	6	6

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					•

Description

Loads the memory location whose address is in HL, with *r* (any of the registers A, B, C, D, E, H, or L).

LD (HL),BCDE

Opcode	Instruction	Operation
DD 1B	LD (HL),BCDE	(HL) = E (HL + 1) = D (HL + 2) = C (HL + 3) = B

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	18	n/a	n/a
Rabbit 5000	18	18	16

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					•

Description

Loads the memory location whose address is in HL with BCDE.

LD (HL),JKHL

Opcode	Instruction	Operation
FD 1B	LD (HL),JKHL	(HL) = L (HL + 1) = H (HL + 2) = JK_{low} (HL + 3) = JK_{high}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	18	n/a	n/a
Rabbit 5000	18	18	16

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					•

Description

Loads the memory location whose address is in HL with JKHL.

LD (HL+*d*),HL

Opcode	Instruction	Operation
DD F4 *d*	LD (HL+*d*),HL	(HL + *d*) = L (HL + *d* + 1) = H

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	13	n/a	n/a
Rabbit 5000	14	13	12

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					•

Description

Loads the memory location whose address is the sum of HL and the 8-bit signed displacement *d* with HL.

LD (IX+*d*),HL

Opcode	Instruction	Operation
F4 *d*	LD (IX+*d*),HL	(IX + *d*) = L (IX + *d* + 1) = H

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	11	n/a	n/a
Rabbit 5000	12	12	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					•

Description

Loads the memory location whose address is the sum of IX and the 8-bit signed displacement *d* with HL.

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

Opcode	Instruction	Operation
DD 36 *d n*	LD (IX+*d*),*n*	(IX + *d*) = *n*

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	11	n/a	n/a
Rabbit 5000	12	10	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					•

Description

Loads the memory location whose address is the sum of IX and the 8-bit signed displacement *d* with the 8-bit constant *n*.

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

Opcode	Instruction	Operation
——	**LD (IX+*d*),*r***	**(IX + *d*) = *r***
DD 77 *d*	LD (IX+*d*),A	(IX + *d*) = A
DD 70 *d*	LD (IX+*d*),B	(IX + *d*) = B
DD 71 *d*	LD (IX+*d*),C	(IX + *d*) = C
DD 72 *d*	LD (IX+*d*),D	(IX + *d*) = D
DD 73 *d*	LD (IX+*d*),E	(IX + *d*) = E
DD 74 *d*	LD (IX+*d*),H	(IX + *d*) = H
DD 75 *d*	LD (IX+*d*),L	(IX + *d*) = L

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	10	n/a	n/a
Rabbit 5000	11	9	9

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					•

Description

Loads the memory location whose address is the sum of IX and the 8-bit signed displacement *d* with *r* (any of the registers A, B, C, D, E, H, or L).

LD (IX+d),BCDE
LD (IX+d),JKHL

Opcode	Instruction	Operation
DD CF *d*	LD (IX+*d*),BCDE	(IX + *d*) = E (IX + *d* + 1) = D (IX + *d* + 2) = C (IX + *d* + 3) = B
FD CF *d*	LD (IX+*d*),JKHL	(IX + *d*) = L (IX + *d* + 1) = H (IX + *d* + 2) = JK_{low} (IX + *d* +3) = JK_{high}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	19	n/a	n/a
Rabbit 5000	20	19	18

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					•

Description

Loads the memory location whose address is the sum of IX and the 8-bit signed displacement *d* with BCDE or JKHL.

LD (IY+d),BCDE
LD (IY+d),JKHL

Opcode	Instruction	Operation
DD DF *d*	LD (IY+d),BCDE	(IY + d) = E (IY + d + 1) = D (IY + d + 2) = C (IY + d + 3) = B
FD DF *d*	LD (IY+d),JKHL	(IY + d) = L (IY + d + 1) = H (IY + d + 2) = JK_{low} (IY + d + 3) = JK_{high}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	19	n/a	n/a
Rabbit 5000	20	19	18

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					•

Description

Loads the memory location whose address is the sum of IY and the 8-bit signed displacement *d* with BCDE or JKHL.

`LD (IY+d),HL`

Opcode	Instruction	Operation
FD F4 *d*	LD (IY+*d*),HL	(IY + *d*) = L (IY + *d* + 1) = H

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	13	n/a	n/a
Rabbit 5000	14	13	12

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
–	–	–	–					•

Description

Loads the memory location whose address is the sum of IY and the 8-bit signed displacement *d* with HL.

`LD (IY+d),n`

Opcode	Instruction	Operation
FD 36 *d n*	LD (IY+*d*),*n*	(IY + *d*) = *n*

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	11	n/a	n/a
Rabbit 5000	12	10	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					•

Description

Loads the memory location whose address is the sum of IY and the 8-bit signed displacement *d* with the 8-bit constant *n*.

`LD (IY+d),r`

Opcode	Instruction	Operation
——	`LD (IY+d),r`	`(IY + d) = r`
FD 77 *d*	LD (IY+*d*),A	(IY + *d*) = A
FD 70 *d*	LD (IY+*d*),B	(IY + *d*) = B
FD 71 *d*	LD (IY+*d*),C	(IY + *d*) = C
FD 72 *d*	LD (IY+*d*),D	(IY + *d*) = D
FD 73 *d*	LD (IY+*d*),E	(IY + *d*) = E
FD 74 *d*	LD (IY+*d*),H	(IY + *d*) = H
FD 75 *d*	LD (IY+*d*),L	(IY + *d*) = L

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	10	n/a	n/a
Rabbit 5000	11	9	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					•

Description

Loads the memory location whose address is the sum of IY and the 8-bit signed displacement *d* with *r* (any of the registers A, B, C, D, E, H, or L).

LD (*mn*),A

Opcode	Instruction	Operation
32 *n* *m*	LD (*mn*),A	(*mn*) = A

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	10	n/a	n/a
Rabbit 5000	10	8	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					•

Description

Loads the memory location whose address is *mn* with A.

`LD (`*`mn`*`),HL`

Opcode	Instruction	Operation
22 *n* *m*	LD (*mn*),HL	(*mn*) = L; (*mn* + 1) = H

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	13	n/a	n/a
Rabbit 5000	13	11	11

Flags				ALTD			IOI/IOE	
`S`	`Z`	`L/V`	`C`	`F`	`R`	`SP`	`S`	`D`
-	-	-	-					•

Description

Loads the memory location whose address is *mn* with HL.

There are two opcodes for “ld (mn),HL” instruction. The assembler generates the shorter one (`22` *`n`* *`m`*).

LD (*mn*),IX
LD (*mn*),IY

Opcode	Instruction	Operation
DD 22 *n* *m*	LD (*mn*),IX	(*mn*) = IX_{low}; (*mn* + 1) = IX_{high}
FD 22 *n* *m*	LD (*mn*),IY	(*mn*) = IY_{low}; (*mn* + 1) = IY_{high}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	15	n/a	n/a
Rabbit 5000	15	13	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					•

Description

Loads the memory location whose address is *mn* with IX or IY.

`LD (mn),ss`

Opcode	Instruction	Operation
——	`LD (mn),ss`	`(mn) = ss`$_{low}$`;` `(mn + 1) = ss`$_{high}$
ED 43 *n m*	LD (*mn*),BC	(*mn*) = C; (*mn* + 1) = B
ED 53 *n m*	LD (*mn*),DE	(*mn*) = E; (*mn* + 1) = D
ED 63 *n m*	LD (*mn*),HL	(*mn*) = L; (*mn* + 1) = H
ED 73 *n m*	LD (*mn*),SP	(*mn*) = SP_{low}; (*mn* + 1) = SP_{high}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	15	n/a	n/a
Rabbit 5000	15	13	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					•

Description

Loads the memory location whose address is *mn* with *ss* (any of the registers BC, DE, HL or SP).

There are two opcodes for "ld (mn),HL" instruction. The assembler generates the shorter one (`22` *n m*). See the instruction LD (mn),HL on the previous page.

LD (*mn*),BCDE
LD (*mn*),JKHL

Opcode	Instruction	Operation
83 *n m*	LD (*mn*),BCDE	(*mn*) = E (*mn* + 1) = D (*mn* + 2) = C (*mn* + 3) = B
84 *n m*	LD (*mn*),JKHL	(*mn*) = L (*mn* + 1) = H (*mn* + 2) = JK_{low} (*mn* + 3) = JK_{high}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	19	n/a	n/a
Rabbit 5000	19	17	17

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					•

Description

Loads the memory location whose address is *mn* with BCDE or JKHL.

LD (*mn*),JK

Opcode	Instruction	Operation
89 *n* *m*	LD (*mn*),JK	(*mn*) = JK

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	13	n/a	n/a
Rabbit 5000	13	11	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					•

Description

Loads the memory location whose address is *mn* with JK.

LD (*pd*+BC),HL

Opcode	Instruction	Operation
—	**LD (*pd*+BC),HL**	**(*pd* + BC) = L** **(*pd* + BC + 1) = H**
ED 07	LD (PW+BC),HL	(PW + BC) = L (PW + BC + 1) = H
ED 17	LD (PX+BC),HL	(PX + BC) = L (PX + BC + 1) = H
ED 27	LD (PY+BC),HL	(PY + BC) = L (PY + BC + 1) = H
ED 37	LD (PZ+BC),HL	(PZ + BC) = L (PZ + BC + 1) = H

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	12	n/a	n/a
Rabbit 5000	13	13	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose address is computed as the sum of *pd* and BC with HL.

The address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If *pd* is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

LD (*pd*+*d*),A

Opcode	Instruction	Operation
—	**LD (*pd*+*d*),A**	**(*pd* + *d*) = A**
8E *d*	LD (PW+*d*),A	(PW + d) = A
9E *d*	LD (PX+*d*),A	(PX + d) = A
AE *d*	LD (PY+*d*),A	(PY + d) = A
BE *d*	LD (PZ+*d*),A	(PZ + d) = A

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	8	n/a	n/a
Rabbit 5000	9	9	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose address is computed as the sum of *pd* and the 8-bit signed displacement *d* with A.

The address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If *pd* is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

LD (*pd*+*d*),BCDE

Opcode	Instruction	Operation
—	**LD (*pd*+*d*),BCDE**	**(*pd*+*d*) = E; (*pd*+*d*+1) = D** **(*pd*+*d*+2) = C; (*pd*+*d*+3) = B**
DD 0F *d*	LD (PW+*d*),BCDE	((PW+*d*) = E; (PW+*d*+1) = D (PW+*d*+2) = C; (PW+*d*+3) = B
DD 1F *d*	LD (PX+*d*),BCDE	((PX+*d*) = E; (PX+*d*+1) = D (PX+*d*+2) = C; (PX+*d*+3) = B
DD 2F *d*	LD (PY+*d*),BCDE	((PY+*d*) = E; (PY+*d*+1) = D (PY+*d*+2) = C; (PY+*d*+3) = B
DD 3F *d*	LD (PZ+*d*),BCDE	((PZ+*d*) = E; (PZ+*d*+1) = D (PZ+*d*+2) = C; (PZ+*d*+3) = B

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	19	n/a	n/a
Rabbit 5000	20	19	18

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose address is computed as the sum of *pd* and the 8-bit signed displacement *d* with BCDE.

The address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If *pd* is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

LD (*pd*+*d*),HL

Opcode	Instruction	Operation
—	**LD (*pd*+*d*),HL**	**(*pd* + *d*) = L** **(*pd* + *d* + 1) = H**
86 *d*	LD (PW+*d*),HL	(PW + *d*) = L (PW + *d* + 1) = H
96 *d*	LD (PX+*d*),HL	(PX + *d*) = L (PX + *d* + 1) = H
A6 *d*	LD (PY+*d*),HL	(PY + *d*) = L (PY + *d* + 1) = H
B6 *d*	LD (PZ+*d*),HL	(PZ + *d*) = L (PZ + *d* + 1) = H

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	11	n/a	n/a
Rabbit 5000	12	12	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose address is computed as the sum of *pd* and the 8-bit signed displacement *d* with HL.

The address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If *pd* is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

LD (*pd*+*d*),JKHL

Opcode	Instruction	Operation
—	**LD (*pd*+*d*),JKHL**	**(*pd* + *d*) = L** **(*pd* + *d* + 1) = H** **(*pd* + *d* + 2) = JK_{low}** **(*pd* + *d* + 3) = JK_{high}**
FD 0F *d*	LD (PW+*d*),JKHL	(PW+*d*) = L; (PW+*d*+1) = H (PW+*d*+2) = JK_{low} (PW+*d*+3) = JK_{high}
FD 1F *d*	LD (PX+*d*),JKHL	(PX+*d*) = L; (PX+*d*+1) = H (PX+*d*+2) = JK_{low} (PX+*d*+3) = JK_{high}
FD 2F *d*	LD (PY+*d*),JKHL	(PY+*d*) = L; (PY+*d*+1) = H (PY+*d*+2) = JK_{low} (PY+*d*+3) = JK_{high}
FD 3F *d*	LD (PZ+*d*),JKHL	(PZ+*d*) = L; (PZ+*d*+1) = H (PZ+*d*+2) = JK_{low} (PZ+*d*+3) = JK_{high}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	19	n/a	n/a
Rabbit 5000	20	19	18

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose address is computed as the sum of *pd* and the 8-bit signed displacement *d* with JKHL.

The address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation. If *pd* is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

LD (*pd*+*d*),ps

Opcode	Instruction	Operation
—	**LD (*pd*+*d*),*ps***	**(*pd*+*d*)=ps_0; (*pd*+*d*+1)=ps_1 (*pd*+*d*+2)=ps_2; (*pd*+*d*+3)=ps_3**
6D 09 *d* 6D 19 *d* 6D 29 *d* 6D 39 *d*	LD (PW+*d*),PW LD (PX+*d*),PW LD (PY+*d*),PW LD (PZ+*d*),PW	(*pd*+d)=PW_0; (*pd*+d+1)=PW_1 (*pd*+d+2)=PW_2; (*pd*+d+3)=PW_3
6D 49 *d* 6D 59 *d* 6D 69 *d* 6D 79 *d*	LD (PW+*d*),PX LD (PX+*d*),PX LD (PY+*d*),PX LD (PZ+*d*),PX	(*pd*+*d*)=PX_0; (*pd*+*d*L+1)=PX_1 (*pd*+*d*+2)=PX_2; (*pd*+*d*+3)=PX_3
6D 89 *d* 6D 99 *d* 6D A9 *d* 6D B9 *d*	LD (PW+*d*),PY LD (PX+*d*),PY LD (PY+*d*),PY LD (PZ+*d*),PY	(*pd*+*d*)=PY_0; (*pd*+*d*+1)=PY_1 (*pd*+*d*+2)=PY_2; (*pd*+*d*+3)=PY_3
6D C9 *d* 6D D9 *d* 6D E9 *d* 6D F9 *d*	LD (PW+*d*),PZ LD (PX+*d*),PZ LD (PY+*d*),PZ LD (PZ+*d*),PZ	(*pd*+*d*)=PZ_0; (*pd*+*d*+1)=PZ_1 (*pd*+*d*+2)=PZ_2; (*pd*+*d*+3)=PZ_3

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	19	n/a	n/a
Rabbit 5000	20	19	18

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose address is computed as the sum of *pd* and the 8-bit signed displacement *d* with *ps*.

The address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation. If *pd* is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

LD (*pd*+*d*),rr

Opcode	Instruction	Operation
—	**LD (*pd*+*d*),rr**	**(pd+d) = rrl; (pd+d+1) = rrh**
6D 01 *d*	LD (PW+*d*),BC	(PW+*d*) = C; (PW+*d*+1) = B
6D 11 *d*	LD (PX+*d*),BC	(PX+*d*) = C; (PX+*d*+1) = B
6D 21 *d*	LD (PY+*d*),BC	(PY+*d*) = C; (PY+*d*+1) = B
6D 31 *d*	LD (PZ+*d*),BC	(PZ+*d*) = C; (PZ+*d*+1) = B
6D 41 *d*	LD (PW+*d*),DE	(PW+*d*) = E; (PW+*d*+1) = D
6D 51 *d*	LD (PX+*d*),DE	(PX+*d*) = E; (PX+*d*+1) = D
6D 61 *d*	LD (PY+*d*),DE	(PY+*d*) = E; (PY+*d*+1) = D
6D 71 *d*	LD (PZ+*d*),DE	(PZ+*d*) = E; (PZ+*d*+1) = D
6D 81 *d*	LD (PW+*d*),IX	(PW+*d*)=IX_{low}; (PW+*d*+1)=IX_{high}
6D 91 *d*	LD (PX+*d*),IX	(PX+*d*)=IX_{low}; (PX+*d*+1)=IX_{high}
6D A1 *d*	LD (PY+*d*),IX	(PY+*d*)=IX_{low}; (PY+*d*+1)=IX_{high}
6D B1 *d*	LD (PZ+*d*),IX	(PZ+*d*)=IX_{low}; (PZ+*d*+1)=IX_{high}
6D C1 *d*	LD (PW+*d*),IY	(PW+*d*)=IY_{low}; (PW+*d*+1)=IY_{high}
6D D1 *d*	LD (PX+*d*),IY	(PX+*d*)=IY_{low}; (PX+*d*+1)=IY_{high}
6D E1 *d*	LD (PY+*d*),IY	(PY+*d*)=IY_{low}; (PY+*d*+1)=IY_{high}
6D F1 *d*	LD (PZ+*d*),IY	(PZ+*d*)=IY_{low}; (PZ+*d*+1)=IY_{higH}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	13	n/a	n/a
Rabbit 5000	14	13	12

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose address is computed as the sum of *pd* and the 8-bit signed displacement *d* with `rr` (any of the 16-bit registers BC, DE, IX or IY).

The address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If `pd` is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a "long logical" address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

`LD (`*`pd`*`+HL),A`

Opcode	Instruction	Operation
—	**LD (*pd*+HL),A**	**(*pd* + HL) = A**
8C	LD (PW+HL),A	(PW + HL) = A
9C	LD (PX+HL),A	(PX + HL) = A
AC	LD (PY+HL),A	(PY + HL) = A
BC	LD (PZ+HL),A	(PZ + HL) = A

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	7	n/a	n/a
Rabbit 5000	8	8	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose address is computed as the sum of *pd* and HL with A. HL is considered sign extended to 24 bits.

The address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If *pd* is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

LD (*pd*+HL),BCDE

Opcode	Instruction	Operation
—	**LD (*pd*+HL),BCDE**	**(*pd*+HL) = E; (*pd*+HL+1) = D (*pd*+HL+2) = C; (*pd*+HL+3) = B**
DD 0D	LD (PW+HL),BCDE	(PW+HL) = E; (PW+HL+1) = D (PW+HL+2) = C; (PW+HL+3) = B
DD 1D	LD (PX+HL),BCDE	(PX+HL) = E; (PX+HL+1) = D (PX+HL+2) = C; (PX+HL+3) = B
DD 2D	LD (PY+HL),BCDE	(PY+HL) = E; (PY+HL+1) = D (PY+HL+2) = C; (PY+HL+3) = B
DD 3D	LD (PZ+HL),BCDE	(PZ+HL) = E; (PZ+HL+1) = D (PZ+HL+2) = C; (PZ+HL+3) = B

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	18	n/a	n/a
Rabbit 5000	19	19	17

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose address is computed as the sum of *pd* and HL with BCDE. HL is considered sign extended to 24 bits.

The address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If *pd* is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

LD (*pd*+HL),JKHL

Opcode	Instruction	Operation
—	**LD (*pd*+HL),JKHL**	**(*pd*+HL) = L; (*pd*+HL+1) = H (*pd*+HL+2) = JK_{low} (*pd*+HL+3) = JK_{high}**
FD 0D	LD (PW+HL),JKHL	(PW+HL) = L; (PW+HL+1) = H (PW+HL+2) = JK_{low} (PW+HL+3) = JK_{high}
FD 1D	LD (PX+HL),JKHL	(PX+HL) = L; (PX+HL+1) = H (PX+HL+2) = JK_{low} (PX+HL+3) = JK_{high}
FD 2D	LD (PY+HL),JKHL	(PY+HL) = L; (PY+HL+1) = H (PY+HL+2) = JK_{low} (PY+HL+3) = JK_{high}
FD 3D	LD (PZ+HL),JKHL	(PZ+HL) = L; (PZ+HL+1) = H (PZ+HL+2) = JK_{low} (PZ+HL+3) = JK_{high}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	18	n/a	n/a
Rabbit 5000	19	19	17

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose address is computed as the sum of *pd* and HL with JKHL. HL is considered sign extended to 24 bits.

The address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If *pd* is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

LD (*pd*+HL),*ps*

Opcode	Instruction	Operation
—	**LD (*pd*+HL),*ps***	**(*pd*+HL)=ps_0; (*pd*+HL+1)=ps_1 (*pd*+HL+2)=ps_2; (*pd*+HL+3)=ps_3**
6D 0B 6D 1B 6D 2B 6D 3B	LD (PW+HL),PW LD (PX+HL),PW LD (PY+HL),PW LD (PZ+HL),PW	(*pd*+HL)=PW_0; (*pd*+HL+1)=PW_1 (*pd*+HL+2)=PW_2; (*pd*+HL+3)=PW_3
6D 4B 6D 5B 6D 6B 6D 7B	LD (PW+HL),PX LD (PX+HL),PX LD (PY+HL),PX LD (PZ+HL),PX	(*pd*+HL)=PX_0; (*pd*+HL+1)=PX_1 (*pd*+HL+2)=PX_2; (*pd*+HL+3)=PX_3
6D 8B 6D 9B 6D AB 6D BB	LD (PW+HL),PY LD (PX+HL),PY LD (PY+HL),PY LD (PZ+HL),PY	(*pd*+HL)=PY_0; (*pd*+HL+1)=PY_1 (*pd*+HL+2)=PY_2; (*pd*+HL+3)=PY_3
6D CB 6D DB 6D EB 6D FB	LD (PW+HL),PZ LD (PX+HL),PZ LD (PY+HL),PZ LD (PZ+HL),PZ	(*pd*+HL)=PZ_0; (*pd*+HL+1)=PZ_1 (*pd*+HL+2)=PZ_2; (*pd*+HL+3)=PZ_3

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	18	n/a	n/a
Rabbit 5000	19	19	17

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose address is computed as the sum of *pd* and HL with *ps* (any of the 32-bit registers PW, PX, PY or PZ).

The address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If *pd* is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a “long logical” address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

LD (*pd*+HL),*rr*

Opcode	Instruction	Operation
— 6D 03 6D 13 6D 23 6D 33	**LD (*pd*+HL),BC** LD (PW+HL),BC LD (PX+HL),BC LD (PY+HL),BC LD (PZ+HL),BC	**(*pd*+HL)=C; (*pd*+HL)=B** (PW+HL)=C; (PW+HL+1)=B (PX+HL)=C; (PX+HL+1)=B (PY+HL)=C; (PY+HL+1)=B (PZ+HL)=C; (PZ+HL+1)=B
— 6D 43 6D 53 6D 63 6D 73	**LD (*pd*+HL),DE** LD (PW+HL),DE LD (PX+HL),DE LD (PY+HL),DE LD (PZ+HL),DE	**(*pd*+HL)=E; (*pd*+HL)=D** (PW+HL)=E; (PW+HL+1)=D (PX+HL)=E; (PX+HL+1)=D (PY+HL)=E; (PY+HL+1)=D (PZ+HL)=E; (PZ+HL+1)=D
— 6D 83 6D 93 6D A3 6D B3	**LD (*pd*+HL),IX** LD (PW+HL),IX LD (PX+HL),IX LD (PY+HL),IX LD (PZ+HL),IX	**(*pd*+HL)=IX_{low}; (*pd*+HL)=IX_{high}** (PW+HL)=IX_{low};(PW+HL+1)=IX_{high} (PX+HL)=IX_{low};(PX+HL+1)=IX_{high} (PY+HL)=IX_{low};(PY+HL+1)=IX_{high} (PZ+HL)=IX_{low};(PZ+HL+1)=IX_{high}
— 6D C3 6D D3 6D E3 6D F3	**LD (*pd*+HL),IY** LD (PW+HL),IY LD (PX+HL),IY LD (PY+HL),IY LD (PZ+HL),IY	**(*pd*+HL)=IY_{low}; (*pd*+HL)=IY high** (PW+HL)=IY_{low};(PW+HL+1)=IY_{high} (PX+HL)=IY_{low};(PX+HL+1)=IY_{high} (PY+HL)=IY_{low};(PY+HL+1)=IY_{high} (PZ+HL)=IY_{low};(PZ+HL+1)=IY_{high}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	12	n/a	n/a
Rabbit 5000	13	13	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose address is computed as the sum of *pd* and HL with *rr* (any of the registers BC, DE, IX or IY). HL is considered sign extended to 24 bits.

The address is treated either as a logical address that will be passed through the MMU for translation into a physical address or as a physical address that does not need MMU translation.

If `pd` is 0xFFFFxxxx, i.e., the upper 16 bits are all ones, it represents a logical address. This is called a "long logical" address. Otherwise, it is a physical address with the low 20 bits or 24 bits being significant (depending on the memory available).

LD (SP+HL),BCDE

Opcode	Instruction	Operation
DD FF	LD (SP+HL),BCDE	(SP+HL) = E; (SP+HL+1) = D (SP+HL+2) = C; (SP+HL+3) = B

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	18	n/a	n/a
Rabbit 5000	19	19	19

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose address is the sum of SP and HL with BCDE.

LD (SP+HL),JKHL

Opcode	Instruction	Operation
FD FF	LD (SP+HL),JKHL	(SP+HL) = L; (SP+HL+1) = H (SP+HL+2) = JK_{low} (SP+HL+3) = JK_{high}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	18	n/a	n/a
Rabbit 5000	19	19	17

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose address is the sum of SP and HL with JKHL.

`LD (SP+n),HL`

Opcode	Instruction	Operation
D4 *n*	LD (SP+*n*),HL	(SP + *n*) = L (SP + *n* + 1) = H

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	11	n/a	n/a
Rabbit 5000	12	12	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose address is the sum of SP and the 8-bit unsigned constant *n* with HL.

`LD (SP+n),IX`
`LD (SP+n),IY`

Opcode	Instruction	Operation
DD D4 *n*	LD (SP+*n*),IX	(SP + *n*) = IX_{low} (SP + *n* + 1) = IX_{high}
FD D4 *n*	LP (SP+*n*),IY	(SP + *n*) = IY_{low} (SP + *n* + 1) = IY_{high}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	13	n/a	n/a
Rabbit 5000	14	13	12

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose address is the sum of SP and the 8-bit unsigned constant *n* with IX or IY.

`LD (SP+n),BCDE`
`LD (SP+n),JKHL`

Opcode	Instruction	Operation
DD EF *n*	LD (SP+*n*),BCDE	(SP + *n*) = E (SP + *n* + 1) = D (SP + *n* + 2) = C (SP + *n* + 3) = B
FD EF *n*	LD (SP+*n*),JKHL	(SP + *n*) = L (SP + *n* + 1) = H (SP + *n* + 2) = JK_{low} (SP + *n* + 3) = JK_{high}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	19	n/a	n/a
Rabbit 5000	20	19	18

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose address is the sum of SP and the 8-bit unsigned constant *n* with BCDE or JKHL.

`LD (SP+n),ps`

Opcode	Instruction	Operation
—	**`LD (SP+n),ps`**	**(SP+n)=ps_0; (SP+n+1)=ps_1 (SP+n+2)=ps_2; (SP+n+3)=ps_3**
ED 05 *n*	LD (SP+*n*),PW	(SP+n)=PW_0; (SP+n+1)=PW_1 (SP+n+2)=PW_2; (SP+n+3)=PW_3
ED 15 *n*	LD (SP+*n*),PX	(SP+n)=PX_0; (SP+n+1)=PX_1 (SP+n+2)=PX_2; (SP+n+3)=PX_3
ED 25 *n*	LD (SP+*n*),PY	(SP+n)=PY_0; (SP+n+1)=PY_1 (SP+n+2)=PY_2; (SP+n+3)=PY_3
ED 35 *n*	LD (SP+*n*),PZ	(SP+n)=PZ_0; (SP+n+1)=PZ_1 (SP+n+2)=PZ_2; (SP+n+3)=PZ_3

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	19	n/a	n/a
Rabbit 5000	20	19	18

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose address is the sum of SP and the 8-bit unsigned constant *n* with *ps* (any of the 32-bit registers PW, PX, PY or PZ).

LDD
LDI

Opcode	Instruction	Operation

ED A8	LDD	(DE) = (HL)
		BC = BC - 1
		DE = DE - 1
		HL = HL - 1

ED A0	LDI	(DE) = (HL)
		BC = BC - 1
		DE = DE + 1
		HL = HL + 1

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	10	n/a	n/a
Rabbit 5000	10	10	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	•	-					•

Description

- `LDD:` Loads the memory location whose address is DE with the data at the address in HL. Then it decrements the value in BC, DE, and HL.
- `LDI:` Loads the memory location whose address is DE with the data at the address in HL. Then the value in BC is decremented and the value in DE and HL is incremented.

If either instruction is prefixed by IOI or IOE, the destination will be in the specified I/O space. Add 1 clock for each iteration 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.

LDDR
LDIR

Opcode	Instruction	Operation

ED B8	LDDR	(DE) = (HL)
		BC = BC - 1
		DE = DE - 1
		HL = HL - 1
		repeat while { BC != 0 }

ED B0	LDIR	(DE) = (HL)
		BC = BC - 1
		DE = DE + 1
		HL = HL + 1
		repeat while { BC != 0 }

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	6+7i	n/a	n/a
Rabbit 5000	6+7i	6+7i	4+7i

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	•	-					•

Description

- `LDDR`: BC holds the count, which is the number of bytes that will be moved from the source address in HL to the destination address in DE. If the count starts at zero, the number of bytes that will be moved is 65536. After each byte is copied, BC, DE and HL are decremented. The instruction repeats until BC reaches zero.
- `LDIR`: BC holds the count, which is the number of bytes that will be moved from the source address in HL to the destination address in DE. If the count starts at zero, the number of bytes that will be moved is 65536. After each byte is copied, BC is decremented and DE and HL are incremented. The instruction repeats until BC reaches zero.

If either of these instructions is prefixed by IOI or IOE, the destination will be in the specified I/O space. If the prefix is IOI, add 1 clock for each iteration. 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, which ends the block copy.

Interrupts can occur between different repeats (after the registers have been updated), 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.

LDDSR
LDISR

Opcode	Instruction	Operation

ED 98	LDDSR	(DE) = (HL)
		BC = BC - 1
		HL = HL - 1
		repeat while BC != 0

ED 90	LDISR	(DE) = (HL)
		BC = BC - 1
		HL = HL + 1
		repeat while BC != 0

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 3000A/4000	6+7i	n/a	n/a
Rabbit 5000	6+7i	6+7i	4+7i

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	•	-					•

Description

- `LDDSR`: BC holds the count, which is the number of bytes that will be moved from the source address in HL to the destination address in DE. If the count starts at zero, the number of bytes that will be moved is 65536. After each byte is copied, BC and HL are decremented. The instruction repeats until BC reaches zero.
- `LDISR`: BC holds the count, which is the number of bytes that will be moved from the source address in HL to the destination address in DE. If the count starts at zero, the number of bytes that will be moved is 65536. After each byte is copied, BC is decremented and HL is incremented. The instruction repeats until BC reaches zero.

These instructions are only useful when prefixed by IOI or IOE. If the prefix is IOI (internal I/O), add 1 clock for each iteration. 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, which ends the block copy.

Interrupts can occur between different repeats (after the registers have been updated), 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.

Load Far

4000, 5000

LDF A,(*lmn*)

Opcode	Instruction	Operation
9A *n m l*	LDF A,(*lmn*)	A = (*lmn*)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	11	n/a	n/a
Rabbit 5000	11	9	7

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads A with the data whose physical address is the 24-bit constant *lmn*.

LDF HL,(*lmn*)

Opcode	Instruction	Operation

92 *n m l*	LDF HL,(*lmn*)	L = (*lmn*)
		H = (*lmn* + 1)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	13	n/a	n/a
Rabbit 5000	13	11	9

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads HL with the data whose physical address is the 24-bit constant *lmn*.

LDF BCDE,(*lmn*)
LDF JKHL,(*lmn*)

Opcode	Instruction	Operation

DD 0A *n m l*	LDF BCDE,(*lmn*)	E = (*lmn*)
		D = (*lmn* + 1)
		C = (*lmn* + 2)
		B = (*lmn* + 3)

FD 0A *n m l*	LDF JKHL,(*lmn*)	L = (*lmn*)
		H = (*lmn* + 1)
		K = (*lmn* +2)
		J = (*lmn* + 3)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	19	n/a	n/a
Rabbit 5000	19	15	15

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads BCDE or JKHL with the data whose physical address is the 24-bit constant *lmn*.

LDF *pd*,(*lmn*)

Opcode	Instruction	Operation
—	**LDF pd,(*lmn*)**	**ps_0 = (lmn)** **ps_1 = (lmn + 1)** **ps_2 = (lmn + 2)** **ps_3 = (lmn + 3)**
ED 08 *n m l*	LDF PW,(*lmn*)	PW_0 = (*lmn*) PW_1 = (*lmn* + 1) PW_2 = (*lmn* + 2) PW_3 = (*lmn* + 3)
ED 18 *n m l*	LDF PX,(*lmn*)	PX_0 = (*lmn*) PX_1 = (*lmn* + 1) PX_2 = (*lmn* + 2) PX_3 = (*lmn* + 3)
ED 28 *n m l*	LDF PY,(*lmn*)	PY_0 = (*lmn*) PY_1 = (*lmn* + 1) PY_2 = (*lmn* + 2) PY_3 = (*lmn* + 3)
ED 38 *n m l*	LDF PZ,(*lmn*)	PZ_0 = (*lmn*) PZ_1 = (*lmn* + 1) PZ_2 = (*lmn* + 2) PZ_3 = (*lmn* + 3)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	19	n/a	n/a
Rabbit 5000	19	15	15

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads pd (any of the 32-bit registers PW, PX, PY or PZ) with the data whose physical address is the 24-bit constant *lmn*.

`LDF rr,(lmn)`

Opcode	Instruction	Operation
—	**LDF *rr*,(*lmn*)**	**rr_{low} = (*lmn*)** **rr_{high} = (*lmn* + 1)**

ED 0A *n m l*	LDF BC,(*lmn*)	C = (*lmn*)
		B = (*lmn* + 1)

ED 1A *n m l*	LDF DE,(*lmn*)	E = (*lmn*)
		D = (*lmn* + 1)

ED 2A *n m l*	LDF IX,(*lmn*)	IX_{low} = (*lmn*) IX_{high} = (*lmn* + 1)
ED 3A *n m l*	LDF IY,(*lmn*)	IY_{low} = (*lmn*) IY_{high} = (*lmn* + 1)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	15	n/a	n/a
Rabbit 5000	15	11	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads BC, DE IX or IY with the data whose physical address is the 24-bit constant *lmn*.

The ALTD prefix does not apply to the “LDF IX,(*lmn*)” or “LDF IY,(*lmn*)” instructions.

Load Far 4000, 5000

`LDF (lmn),A`

Opcode	Instruction	Operation
8A *n m l*	LDF (*lmn*),A	(*lmn*) = A

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	12	n/a	n/a
Rabbit 5000	12	10	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose physical address is the 24-bit constant *lmn* with A.

LDF (*lmn*),HL

Opcode	Instruction	Operation
82 *n m l*	LDF (*lmn*),HL	(*lmn*) = L (*lmn* + 1) = H

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	15	n/a	n/a
Rabbit 5000	13	11	9

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose physical address is the 24-bit constant *lmn* with HL.

Load Far 4000, 5000

LDF (*lmn*),BCDE
LDF (*lmn*),JKHL

Opcode	Instruction	Operation
DD 0B *n m l*	LDF (*lmn*),BCDE	(lmn) = E (lmn + 1) = D (lmn + 2) = C (lmn + 3) = B
FD 0B *n m l*	LDF (*lmn*),JKHL	(lmn) = L (lmn + 1) = H (lmn + 2) = JK_{low} (lmn + 3) = JK_{high}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	23	n/a	n/a
Rabbit 5000	23	19	19

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose physical address is the 24-bit constant *lmn* with BCDE or JKHL.

LDF (*lmn*),*ps*

Opcode	Instruction	Operation
—	**LDF (*lmn*),*ps***	**(lmn) = ps_0** **(lmn + 1) = ps_1** **(lmn + 2) = ps_2** **(lmn + 3) = ps_3**
ED 09 *n m l*	LDF (*lmn*),PW	(*lmn*) = PW_0 (*lmn* + 1) = PW_1 (*lmn* + 2) = PW_2 (*lmn* + 3) = PW_3
ED 19 *n m l*	LDF (*lmn*),PX	(*lmn*) = PX_0 (*lmn* + 1) = PX_1 (*lmn* + 2) = PX_2 (*lmn* + 3) = PX_3
ED 29 *n m l*	LDF (*lmn*),PY	(*lmn*) = PY_0 (*lmn* + 1) = PY_1 (*lmn* + 2) = PY_2 (*lmn* + 3) = PY_3
ED 39 *n m l*	LDF (*lmn*),PZ	(*lmn*) = PZ_0 (*lmn* + 1) = PZ_1 (*lmn* + 2) = PZ_2 (*lmn* + 3) = PZ_3

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	23	n/a	n/a
Rabbit 5000	23	19	19

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose physical address is the 24-bit constant *lmn* with ps (any of the 32-bit registers PW, PX, PY or PZ).

LDF (*lmn*),*rr*

Opcode	Instruction	Operation
—	**LDF (*lmn*),*rr***	**(*lmn*) = rr_{low}** **(*lmn* + 1) = rr_{high}**
ED 0B *n m l*	LDF (*lmn*),BC	(*lmn*) = C (*lmn* + 1) = B
ED 1B *n m l*	LDF (*lmn*),DE	(*lmn*) = E (*lmn* + 1) = D
ED 2B *n m l*	LDF (*lmn*),IX	(*lmn*) = IX_{low} (*lmn* + 1) = IX_{high}
ED 3B *n m l*	LDF (*lmn*),IY	(*lmn*) = IY_{low} (*lmn* + 1) = IY_{high}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	17	n/a	n/a
Rabbit 5000	17	13	13

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads the memory location whose physical address is the 24-bit constant *lmn* with BC, DE, IX or IY.

LDL *pd*,DE

Opcode	Instruction	Operation
—	**LDL *pd*,DE**	***pd* = {FFFF,DE}**
DD 8F	LDL PW,DE	PW_0 = E PW_1 = D PW_2 = FF; PW_3 = FF
DD 9F	LDL PX,DE	PX_0 = E PX_1 = D PX_2 = FF; PX_3 = FF
DD AF	LDL PY,DE	PY_0 = E PY_1 = D PY_2 = FF; PY_3 = FF
DD BF	LDL PZ,DE	PZ_0 = E PZ_1 = D PZ_2 = FF; PZ_3 = FF

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads the lower 16 bits of *pd* (any of the 32-bits registers PW, PX, PY or PZ) with DE. The upper word of *pd* is loaded with 0xFFFF.

LDL *pd*,HL

Opcode	Instruction	Operation
—	**LDL *pd*,HL**	***pd* = {FFFF,HL}**
FD 8F	LDL PW,HL	PW_0 = L PW_1 = H PW_2 = FF; PW_3 = FF
FD 9F	LDL PX,HL	PX_0 = L PX_1 = H PX_2 = FF; PX_3 = FF
FD AF	LDL PY,HL	PY_0 = L PY_1 = H PY_2 = FF; PY_3 = FF
FD BF	LDL PZ,HL	PZ_0 = L PZ_1 = H PZ_2 = FF; PZ_3 = FF

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads the low word of *pd* (any of the 32-bit registers PW, PX, PY or PZ) with HL. Loads the high word with 0xFFFF.

LDL *pd*,IX

Opcode	Instruction	Operation
—	**LDL *pd*,IX**	***pd* = {FFFF,IX}**
DD 8C	LDL PW,IX	PW_0 = IX_{low} PW_1 = IX_{high} PW_2 = FF; PW_3 = FF
DD 9C	LDL PX,IX	PX_0 = IX_{low} PX_1 = IX_{high} PX_2 = FF; PX_3 = FF
DD AC	LDL PY,IX	PY_0 = IX_{low} PY_1 = IX_{high} PY_2 = FF; PY_3 = FF
DD BC	LDL PZ,IX	PZ_0 = IX_{low} PZ_1 = IX_{high} PZ_2 = FF; PZ_3 = FF

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads the low-order word of *pd* (any of the 32-bit registers PW, PX, PY or PZ) with IX. Loads the high-order word with 0xFFFF.

LDL *pd*,IY

Opcode	Instruction	Operation
—	**LDL *pd*,IY**	***pd* = {FFFF,IY}**
FD 8C	LDL PW,IY	PW_0 = IY_{low} PW_1 = IY_{high} PW_2 = FF; PW_3 = FF
FD 9C	LDL PX,IY	PX_0 = IY_{low} PX_1 = IY_{high} PX_2 = FF; PXW_3 = FF
FD AC	LDL PY,IY	PY_0 = IY_{low} PY_1 = IY_{high} PY_2 = FF; PY_3 = FF
FD BC	LDL PZ,IY	PZ_0 = IY_{low} PZ_1 = IY_{high} PZ_2 = FF; PZ_3 = FF

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads the low-order word of *pd* (any of the 32-bit registers PW, PX, PY or PZ) with IY. Loads the high-order word with 0xFFFF.

Load Logical

LDL *pd,mn*

Opcode	Instruction	Operation
—	**LDL *pd,mn***	***pd* = {FFFF,*mn*}**
ED 0D *n m*	LDL PW,*mn*	PW_0 = *n* PW_1 = *m* PW_2 = FF; PW_3 = FF
ED 1D *n m*	LDL PX,*mn*	PX_0 = *n* PX_1 = *m* PX_2 = FF; PX_3 = FF
ED 2D *n m*	LDL PY,*mn*	PY_0 = *n* PY_1 = *m* PY_2 = FF; PY_3 = FF
ED 3D *n m*	LDL PZ,*mn*	PZ_0 = *n* PZ_1 = *m* PZ_2 = FF; PZ_3 = FF

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	8	n/a	n/a
Rabbit 5000	8	6	4

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads the low-order word of *pd* (any of the 32-bit registers PW, PX, PY or PZ) with the 16-bit constant *mn*. Loads the high-order word with 0xFFFF.

Load Logical

4000, 5000

LDL *pd*,(SP+*n*)

Opcode	Instruction	Operation
—	**LDL pd,(SP+*n*)**	**pd_0 = (SP + *n*)** **pd_1 = (SP + *n* + 1)** **pd_2 = FF; pd_3 = FF**
ED 03	LDL PW,(SP+*n*)	PW_0 = (SP + *n*) PW_1= (SP + *n* +1) PW_2 = FF; PW_3 = FF
ED 13	LDL PX,(SP+*n*)	PX_0 = (SP + *n*) PX_1 = (SP + *n* + 1) PX_2 = FF; PX_3 = FF
ED 23	LDL PY,(SP+*n*)	PY_0 = (SP + *n*) PY_1 = (SP + *n* + 1) PY_2 = FF; PY_3 = FF
ED 33	LDL PZ,(SP+*n*)	PZ_0 = (SP + *n*) PZ_1 = (SP + *n* + 1) PZ_2 = FF; PZ_3 =F F

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	11	n/a	n/a
Rabbit 5000	12	11	10

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads the low-order word of *pd* (any of the 32-bit registers PW, PX, PY or PZ) with the data whose address in the sum of SP and the 8-bit unsigned constant *n*. Loads the high-order word with 0xFFFF.

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

Opcode	Instruction	Operation

ED 6C	LDP HL,(HL)	L = (HL)
		H = (HL + 1)
		(Addr[19:16] = A[3:0])

DD 6C	LDP HL,(IX)	L = (IX)
		H = (IX + 1)
		(Addr[19:16] = A[3:0])

FD 6C	LDP HL,(IY)	L = (IY)
		H = (IY + 1)
		(Addr[19:16] = A[3:0])

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	10	n/a	n/a
Rabbit 5000	10	10	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	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 0x*n*,0xFFFF, you will get the bytes located at 0x*n*, 0xFFFF and 0xn,0x0000 instead of 0x*n*,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	Operation

ED 6D *n* *m*	LDP HL,(*mn*)	L = (*mn*)
		H = (*mn* + 1)
		(Addr[19:16] = A[3:0])

DD 6D *n* *m*	LDP IX,(*mn*)	IX_{low} = (*mn*) IX_{high} = (*mn* + 1); (Addr[19:16] = A[3:0])
FD 6D *n* *m*	LDP IY,(*mn*)	IY_{low} = (*mn*) IY_{high} = (*mn* + 1); (Addr[19:16] = A[3:0])

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	13	n/a	n/a
Rabbit 5000	13	11	9

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

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 wraps around and is 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 0x*n*,0xFFFF, you will get the bytes located at 0x*n*, 0xFFFF and 0x*n*,0x0000 instead of 0x*n*,0xFFFF and 0x(*n*+1)0x0000 as you might expect. Therefore, do not use LDP at any physical address ending in 0xFFFF.

Load Physical

2000, 3000, 4000, 5000

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

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	12	n/a	n/a
Rabbit 5000	12	12	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	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 0x*n*,0xFFFF, you will get the bytes located at 0x*n*, 0xFFFF and 0x*n*,0x0000 instead of 0x*n*,0xFFFF and 0x(*n*+1),0x0000 as you might expect. Therefore, do not use LDP at any physical address ending in 0xFFFF.

Load Physical

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

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	15	n/a	n/a
Rabbit 5000	15	13	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

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 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 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.

Long Jump 2000, 3000, 4000, 5000

`LJP x,mn`

Opcode	Instruction	Operation

C7 *n m x*	LJP x,*mn*	XPC = *x*
		PC = *mn*

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	10	n/a	n/a
Rabbit 5000	10	8	6

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	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 constant, *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.

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

See Also: `SJP label`

Far Call 4000, 5000

`LLCALL lxpc,mn`

Opcode	Instruction	Operation
8F *n m xpl xph*	LLCALL l*xpc*,*mn*	(SP-1) = $LXPC_{high}$ (SP-2) = $LXPC_{low}$ (SP-3) = PC_{high} (SP-4) = PC_{low} $LXPC_{low}$ = xpl $LXPC_{high}$ = xph PC = mn SP = SP-4

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	24	n/a	n/a
Rabbit 5000	25	21	21

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

This instruction is similar to the LCALL[1] instruction in that it transfers program execution to the subroutine address specified by the 16-bit operand *mn* and allows calls to be made to a computed address in extended memory. The LLCALL instruction uses the 12-bit LXPC of the Rabbit 4000 or 5000 processor instead of the 8-bit XPC of earlier Rabbit processors. Note that the value of LXPC and consequently the address space defined by the LXPC is dynamically changed with the LLCALL instructions.

In the LLCALL instruction, first LXPC is pushed onto the stack, high-order byte first, then the low-order byte. Next, PC is pushed onto the stack, high-order byte first, then the low-order byte. Then LXPC is loaded with the 16-bit value *lxpc* (its 4 most significant bits are ignored) and the PC is loaded with the 16-bit value *mn*. SP is then updated.

Alternate Forms

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

```
LLCALL x,label
LLCALL x:label
LLCALL x:mn
```

The parameter “label” is user-defined. The colon is equivalent to the comma as a delimiter.

1. Avoid mixing LCALL and LLCALL instructions. When LCALL pushes the XPC, it also clears the upper bits of the LXPC.

LLCALL (JKHL)

Opcode	Instruction	Operation
ED FA	LLCALL (JKHL)	(SP-1) = $LXPC_{high}$ (SP-2) = $LXPC_{low}$ (SP-3) = PC_{high} (SP-4) = PC_{low} PC = HL LXPC = JK SP = SP-4

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	19	n/a	n/a
Rabbit 5000	20	20	18

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

This instruction is similar to the LCALL[1] instruction in that it transfers program execution to the subroutine address specified by the 16-bit constant *mn* and allows calls to made to a computed address in extended memory. The LLCALL instruction uses the 12-bit LXPC of the Rabbit 4000 or 5000 processor instead of the 8-bit XPC of earlier Rabbit processors. Note that the value of LXPC and consequently the address space defined by the LXPC is dynamically changed with the LLCALL instructions.

In the LLCALL instruction, first LXPC is pushed onto the stack, high-order byte first, then the low-order byte. Next, PC is pushed onto the stack, high-order byte first, then the low-order byte. Then PC is loaded with the data in HL and XPC is loaded with the data in JK. SP is then updated.

1. Avoid mixing LCALL and LLCALL instructions. When LCALL pushes the XPC, it also clears the upper bits of the LXPC.

Far Jump

`LLJP cc,lxpc,mn`

Opcode	Instruction	Operation
—	**LLJP *cc*, *lxpc*,*mn***	**if {*cc*}** **XPC_{low} = $lxpc_{low}$** **XPC_{high} = $lxpc_{high}$** **PC = *mn***
ED C2 *n m xpl xph*	LLJP NZ, *lxpc*,*mn*	if {NZ} $XPC_{low} = lxpc_{low}$ $XPC_{high} = lxpc_{high}$ PC = *mn*
ED CA *n m xpl xph*	LLJP Z, *lxpc*,*mn*	if {Z} $XPC_{low} = lxpc_{low}$ $XPC_{high} = lxpc_{high}$ PC = *mn*
ED D2 *n m xpl xph*	LLJP NC, *lxpc*,*mn*	if {NC} $XPC_{low} = xpc_{low}$ $XPC_{high} = xpc_{high}$ PC = *mn*
ED DA *n m xpl xph*	LLJP C, *lxpc*,*mn*	if {C} $XPC_{low} = lxpc_{low}$ $XPC_{high} = lxpc_{high}$ PC = *mn*

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	14	n/a	n/a
Rabbit 5000	14	10	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

If condition *cc* is true then program execution is transferred to the memory location specified by the 16-bit constant, *mn*. A jump can be made to a computed address in extended memory by loading the 12-bit XPC with the 16-bit constant *lxpc* (the 4 most significant bits of `lxpc` are discarded). Note that the value of the 12-bit XPC and consequently the address space defined by it is dynamically changed with this instruction.

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

Far Jump

4000, 5000

LLJP *cx,lxpc,mn*

Opcode	Instruction	Operation
—	**LLJP *cx*, *lxpc*,*mn***	**if {*cx*}** **XPC_{low} = $lxpc_{low}$** **XPC_{high} = $lxpc_{high}$** **PC = *mn***
ED A2 *n m xpl xph*	LLJP GT, *lxpc*,*mn*	if {GT} XPC_{low} = $lxpc_{low}$ XPC_{high} = $lxpc_{high}$ PC = *mn*
ED AA *n m xpl xph*	LLJP GTU, *lxpc*,*mn*	if {GTU} XPC_{low} = $lxpc_{low}$ XPC_{high} = $lxpc_{high}$ PC = *mn*
ED B2 *n m xpl xph*	LLJP LT, *lxpc*,*mn*	if {LT} XPC_{low} = $lxpc_{low}$ XPC_{high} = $lxpc_{high}$ PC = *mn*
ED BA *n m xpl xph*	LLJP V, *lxpc*,*mn*	if {V} XPC_{low} = $lxpc_{low}$ XPC_{high} = $lxpc_{high}$ PC = *mn*

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	14	n/a	n/a
Rabbit 5000	14	10	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

If condition *cx* is true then program execution is transferred to the memory location specified by the 16-bit constant, *mn*. A jump can be made to a computed address in extended memory by loading the 12-bit XPC with the 16-bit constant *lxpc* (the 4 most significant bits of *lxpc* are discarded). Note that the value of the 12-bit XPC and consequently the address space defined by it is dynamically changed with this instruction.

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

Far Jump 4000, 5000

`LLJP` *`lxpc,mn`*

Opcode	Instruction	Operation
87 *n m xpl xph*	LLJP *lxpc,mn*	$XPC_{low} = lxpc_{low}$ $XPC_{high} = lxpc_{high}$ PC = *mn*

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	12	n/a	n/a
Rabbit 5000	12	8	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Program execution is transferred to the memory location specified by the 16-bit constant, *mn*. A jump can be made to a computed address in extended memory by loading the 12-bit XPC with the 16-bit constant *lxpc* (the 4 most significant bits of `lxpc` are discarded). Note that the value of the 12-bit XPC and consequently the address space defined by it is dynamically changed with this instruction.

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

`LLRET`

Opcode	Instruction	Operation
ED 8B	LLRET	PC_{low} = (SP) PC_{high} = (SP + 1) XPC_{low} = (SP + 2) XPC_{high} = (SP + 3) SP = SP + 4

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	14	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

The `LLRET` instruction is used to return from an `LLCALL` operation. It transfers execution from a subroutine to the calling program by popping PC and the XPC from the stack.

The low-order byte of PC is loaded with the data whose address is SP and the high-order byte of PC is loaded with the data whose address is SP+1. Then, the low-order byte of XPC is loaded with the data whose address is SP+2 and the high-order byte of XPC is loaded with the data whose address is SP+3. Finally, the value in SP is incremented by 4.

Long Return

2000, 3000, 4000, 5000

LRET

Opcode	Instruction	Operation

ED 45	LRET	PC_{low} = (SP)
		PC_{high} = (SP + 1)
		XPC = (SP + 2)
		SP = SP + 3

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	13	n/a	n/a
Rabbit 5000	13	11	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

The `LRET` instruction is used to return from an `LCALL` operation. It transfers execution from a subroutine to the calling program by popping PC and the XPC from the stack.

First, the low-order byte of PC is loaded with the data whose address is SP. Next, the high-order byte of PC is loaded with the data whose address is SP+1. Then, XPC is loaded with the data whose address is SP+2. Finally the value in SP is incremented by 3.

LSDDR
LSIDR

Opcode	Instruction	Operation

ED D8	LSDDR	(DE) = (HL)
		BC = BC - 1
		DE = DE - 1
		repeat while BC != 0

ED D0	LSIDR	(DE) = (HL)
		BC = BC - 1
		DE = DE + 1
		repeat while BC != 0

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 3000A/4000	6+7i	n/a	n/a
Rabbit 5000	6+7i	6+7i	4+7i

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-				•	

Description

- `LSDDR`: BC holds the count, which is the number of bytes that will be copied from the source address in HL to the destination address in DE. If the count starts at zero, the number of bytes that will be moved is 65536. After each byte is copied, BC and DE are decremented and HL remains unchanged. The instruction repeats until BC reaches zero.
- `LSIDR`: BC holds the count, which is the number of bytes that will be copied from the source address in HL to the destination address in DE. If the count starts at zero, the number of bytes that will be moved is 65536. After each byte is copied, BC is decremented and DE is incremented. HL remains unchanged. The instruction repeats until BC reaches zero.

If either of these instructions is prefixed by IOI or IOE, the source will be in the specified I/O space. If the prefix is IOI (internal I/O), add 1 clock for each iteration. 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, which ends the block copy.

Interrupts can occur between different repeats (after the registers have been updated), 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.

LSDR
LSIR

Opcode	Instruction	Operation

ED F8	LSDR	(DE)=(HL)
		BC = BC - 1
		DE = DE - 1
		HL = HL - 1
		repeat while BC != 0

ED F0	LSIR	(DE)=(HL)
		BC = BC - 1
		DE = DE + 1
		HL = HL + 1
		repeat while BC != 0

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 3000A/4000	6+7i	n/a	n/a
Rabbit 5000	6+7i	6+7i	4+7i

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-				•	

Description

- `LSDR`: BC holds the count, which is the number of bytes that will be moved from the source address in HL to the destination address in DE. If the count starts at zero, the number of bytes that will be moved is 65536. After each byte is copied, BC, DE and HL are decremented. The instruction repeats until BC reaches zero.
- `LSIR`: BC holds the count, which is the number of bytes that will be moved from the source address in HL to the destination address in DE. If the count starts at zero, the number of bytes that will be moved is 65536. After each byte is copied, BC is decremented and DE and HL are incremented. The instruction repeats until BC reaches zero.

If either of these instructions is prefixed by IOI or IOE, the source will be in the specified I/O space. If the prefix is IOI, add 1 clock for each iteration. 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, which ends the block copy.

Interrupts can occur between different repeats (after the registers have been updated), 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.

MUL

Opcode	Instruction	Operation
F7	MUL	HL:BC = BC • DE

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	12	n/a	n/a
Rabbit 5000	12	12	12

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

A signed multiplication operation is performed on the 16-bit binary integers in the BC and DE registers. The signed 32-bit result is loaded 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
```

Multiply Unsigned **4000, 5000**

MULU

Opcode	Instruction	Operation
A7	MULU	HL:BC = BC • DE (unsigned)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	12	n/a	n/a
Rabbit 5000	12	12	12

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

An unsigned multiplication operation is performed on the 16-bit binary integers in the BC and DE registers. The unsigned 32-bit result is loaded in HL (bits 31 through 16) and BC (bits 15 through 0).

Examples:

```
LD BC, 0FFFFh       ; BC gets 65,535
LD DE, 0FFFFh       ; DE gets 65,535
MULU                ; HL|BC = 4,294,836,225 HL gets 0xFFFE, BC gets 0x0001

LD BC, 0FFFFh       ; BC gets 65,535
LD DE, 00001h       ; DE gets 1
MULU                ; HL|BC = 65,535, HL gets 0x0000, BC gets 0xFFFF
```

NEG

Opcode	Instruction	Operation
ED 44	NEG	A = 0 - A

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	13	11	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•			

Description

Subtracts A from zero and stores the result in A.

Negate

`NEG BCDE`
`NEG JKHL`

Opcode	Instruction	Operation
DD 4D	NEG BCDE	BCDE = 0 - BCDE
FD 4D	NEG JKHL	JKHL = 0 - JKHL

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•			

Description

Subtracts BCDE or JKHL from zero and stores the result in BCDE or JKHL.

Negate 4000, 5000

NEG HL

Opcode	Instruction	Operation
4D	NEG HL	HL = 0 - HL

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•			

Description

Subtracts HL from zero and stores the result in HL.

No Operation 2000, 3000, 4000, 5000

NOP

Opcode	Instruction	Operation
00	NOP	No operation

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

No operation is performed during this cycle.

OR A

Opcode	Instruction	Operation
B7	OR A	A = A | A

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•			

Description

Performs a bitwise OR operation between A and A. All of the flags are affected and A remains unchanged.

Example

The “OR A” operation results in the following:

If A = 0x7F, S=0; Z=0; L/V=1; C=0.
If A = 0x80, S=1; Z=0; L/V=1; C=0.
If A = 0x00, S=0; Z=1; L/V=0; C=0.

OR HL,DE

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•		•	

Description

Performs a bitwise OR between the data in HL and the data in DE. The result is stored in HL.

OR IX,DE
OR IY,DE

Opcode	Instruction	Operation
DD EC	OR IX,DE	IX = IX | DE
FD EC	OR IY,DE	IY = IY | DE

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•				

Description

Performs a bitwise OR operation between DE and IX or IY.

The result is stored in IX or IY.

OR JKHL,BCDE

Opcode	Instruction	Operation
ED F6	OR JKHL,BCDE	JKHL = JKHL | BCDE

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•			

Description

Performs a bitwise OR operation between JKHL and BCDE and stores the result in JKHL.

Bitwise OR **2000, 3000, 4000, 5000**

`OR n`

Opcode	Instruction	Operation
F6 *n*	OR *n*	A = A | *n*

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•			

Description

Performs a bitwise OR operation between A and the 8-bit constant *n*. The result is stored in A.

The Rabbit 4000/5000 assemblers view “OR A,n” and “OR n” as equivalent instructions.

Bitwise OR

`OR r`

Opcode	Instruction	Operation
——	**OR *r***	**A = A | *r***
B0	OR B	A = A | B
B1	OR C	A = A | C
B2	OR D	A = A | D
B3	OR E	A = A | E
B4	OR H	A = A | H
B5	OR L	A = A | L

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000	2	n/a	n/a

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•			

Description

Performs a bitwise OR operation between A and *r* (any of the 8-bit registers B, C, D, E, H, or L). The result is stored in A.

The opcodes for these instructions are different than the same instructions in the Rabbit 4000 and 5000.

OR *r*

Opcode	Instruction	Operation
—	**OR *r***	**A = A | *r***
7F B0	OR B	A = A | B
7F B1	OR C	A = A | C
7F B2	OR D	A = A | D
7F B3	OR E	A = A | E
7F B4	OR H	A = A | H
7F B5	OR L	A = A | L

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•			

Description

Performs a bitwise OR operation between A and *r* (any of 8-bit registers B, C, D, E, H, L) and stores the result in A.

The Rabbit 4000/5000 assemblers view “OR A,r” and “OR r” as equivalent instructions.

The opcodes for these instructions are different than the same instructions in the Rabbit 2000, 3000 and 3000A.

Example

If the value in A is 0100 1100 and the value in B is 0101 0001, the operation:

```
OR A,B
```

would result in A containing 0101 1101.

Bitwise OR 2000, 3000, 3000A

`OR (HL)`

Opcode	Instruction	Operation
B6	OR (HL)	A = A | (HL)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000	5	n/a	n/a

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•		•	

Description

Performs a bitwise OR operation between A and the data whose address is in HL.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)

Opcode	Instruction	Operation
7F B6	OR (HL)	A = A | (HL)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	7	n/a	n/a
Rabbit 5000	5	5	5

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•		•	

Description

Performs a bitwise OR operation between A and the data whose address is in HL. The result is stored in A.

The Rabbit 4000/5000 assemblers view “OR A,(HL)” and “OR (HL)” as equivalent instructions.

The opcode for these instructions is different than the same instructions in the Rabbit 2000, 3000 and 3000A.

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.

Bitwise OR **2000, 3000, 4000, 5000**

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

Opcode	Instruction	Operation
DD B6 *d*	OR (IX+*d*)	A = A | (IX + *d*)
FD B6 *d*	OR (IY+*d*)	A = A | (IY + *d*)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	9	n/a	n/a
Rabbit 5000	10	9	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•		•	

Description

Performs a bitwise OR between A and the data whose address is

- the sum of the data in IX and the 8-bit signed displacement *d*, or
- the sum of the data in IY and the 8-bit signed displacement *d*.

The result is stored in A.

The Rabbit 4000/5000 assemblers view “OR A,(IX+d)” and “OR (IX+d)” as equivalent instructions. The same is true for “OR A,(IX+d)” and “OR (IX+d).”

Example

If the byte in A is 0100 1100 and the byte in the memory location pointed to by IX+*d* is 1110 0101, the operation:

```
OR (IX+d)
```

would result in A containing 1110 1101.

POP BCDE
POP JKHL

Opcode	Instruction	Operation

DD F1	POP BCDE	E = (SP)
		D = (SP + 1)
		C = (SP + 2)
		B = (SP + 3)
		SP = SP + 4

FD F1	POP JKHL	L = (SP)
		H = (SP + 1)
		K = (SP + 2)
		J = (SP + 3)
		SP = SP + 4

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	13	n/a	n/a
Rabbit 5000	13	13	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Pops BCDE or JKHL from the stack.

POP IP

Opcode	Instruction	Operation

ED 7E	POP IP	IP = (SP)
		SP = SP + 1

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	7	n/a	n/a
Rabbit 5000	7	7	5

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Pops IP from the stack. This is a chained-atomic instruction.

POP IX
POP IY

Opcode	Instruction	Operation
DD E1	POP IX	IX_{low} = (SP) IX_{high} = (SP + 1) SP = SP + 2
FD E1	POP IY	IY_{low} = (SP) IY_{high} = (SP + 1) SP = SP + 2

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	9	n/a	n/a
Rabbit 5000	9	9	7

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Pops IX or IY from the stack.

POP *pd*

Opcode	Instruction	Operation
—	**POP *pd***	**pd_0 = (SP)** **pd_1 = (SP + 1)** **pd_2 = (SP + 2)** **pd_3 = (SP + 3); SP = SP + 4**
ED C1	POP PW	PW_0 = (SP) PW_1 = (SP + 1) PW_2 = (SP + 2) PW_3 = (SP + 3); SP = SP + 4
ED D1	POP PX	PX_0 = (SP) PX_1 = (SP + 1) PX_2 = (SP + 2) PX_3 = (SP + 3); SP = SP + 4
ED E1	POP PY	PY_0 = (SP) PY_1 = (SP + 1) PY_2 = (SP + 2) PY_3 = (SP + 3); SP = SP + 4
ED F1	POP PZ	PZ_0 = (SP) PZ_1 = (SP + 1) PZ_2 = (SP + 2) PZ_3 = (SP + 3); SP = SP + 4

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	13	n/a	n/a
Rabbit 5000	13	13	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Pops *pd* (any of the 32-bit registers PW, PX, PY or PZ) from the stack.

POP SU

Opcode	Instruction	Operation

ED 6E	POP SU	SU = (SP)
		SP = SP + 1

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 3000A/4000	9	n/a	n/a
Rabbit 5000	7	7	5

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	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 is a chained-atomic instruction, meaning that an interrupt cannot take place between this instruction and the instruction following it.

POP *zz*

Opcode	Instruction	Operation
——	**POP *zz***	**zz_{low} = (SP)** **zz_{high} = (SP + 1)** **SP = SP + 2**

F1	POP AF	F = (SP)
		A = (SP + 1)
		SP = SP + 2

C1	POP BC	C = (SP)
		B = (SP + 1)
		SP = SP + 2

D1	POP DE	E = (SP)
		D = (SP + 1)
		SP = SP + 2

E1	POP HL	L = (SP)
		H = (SP + 1)
		SP = SP + 2

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	7	n/a	n/a
Rabbit 5000	7	7	7

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Loads the low-order byte of *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

Opcode	Instruction	Operation
ED 76	PUSH IP	(SP - 1) = IP SP = SP - 1

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	9	n/a	n/a
Rabbit 5000	10	10	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Pushes IP on the stack.

PUSH IX
PUSH IY

Opcode	Instruction	Operation
DD E5	PUSH IX	$(SP - 1) = IX_{high}$ $(SP - 2) = IX_{low}$ $SP = SP - 2$
FD E5	PUSH IY	$(SP - 1) = IY_{high}$ $(SP - 2) = IY_{low}$ $SP = SP - 2$

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	12	n/a	n/a
Rabbit 5000	13	13	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Pushes IX or IY on the stack.

PUSH BCDE
PUSH JKHL

Opcode	Instruction	Operation
DD F5	PUSH BCDE	(SP - 1) = B (SP - 2) = C (SP - 3) = D (SP - 4) = E SP = SP - 4
FD F5	PUSH JKHL	(SP - 1) = JK_{High} (SP - 2) = JK_{Low} (SP - 3) = H (SP - 4) = L SP = SP - 4

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	18	n/a	n/a
Rabbit 5000	19	19	17

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Pushes BCDE or JKHL on the stack.

PUSH *mn*

Opcode	Instruction	Operation
ED A5 *n m*	PUSH *mn*	(SP-1) = *m* (SP-2) = *n* SP = SP - 2

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	15	n/a	n/a
Rabbit 5000	16	14	12

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Pushes the 16-bit constant *mn* on the stack.

PUSH *ps*

Opcode	Instruction	Operation
—	**PUSH *ps***	**(SP - 1) = ps_3** **(SP - 2) = ps_2** **(SP - 3) = ps_1** **(SP - 4) = ps_0** **SP = SP - 4**
ED C5	PUSH PW	(SP - 1) = PW_3; (SP - 2) = PW_2 (SP - 3) = PW_1; (SP - 4) = PW_0 SP = SP - 4
ED D5	PUSH PX	(SP - 1) = PX_3; (SP - 2) = PX_2 (SP - 3) = PX_1; (SP - 4) = PX_0 SP = SP - 4
ED E5	PUSH PY	(SP - 1) = PY_3; (SP - 2) = PY_2 (SP - 3) = PY_1; (SP - 4) = PY_0 SP = SP - 4
ED F5	PUSH PZ	(SP - 1) = PZ_3; (SP - 2) = PZ_2 (SP - 3) = PZ_1; (SP - 4) = PZ_0 SP = SP - 4

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	18	n/a	n/a
Rabbit 5000	19	19	17

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Pushes *ps* (any of the 32-bit registers PW, PX, PY or PZ) on the stack.

PUSH SU

Opcode	Instruction	Operation
ED 66	PUSH SU	(SP - 1) = SU SP = SP - 1

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 3000A/4000	9	n/a	n/a
Rabbit 5000	10	10	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Pushes SU on the stack. This is a chained-atomic instruction, meaning that an interrupt cannot take place between this instruction and the instruction following it.

PUSH *zz*

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	10	n/a	n/a
Rabbit 5000	11	11	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Pushes *zz* (any of the 16-bit registers AF, BC, DE or HL) on the stack.

Read Mode 3000A, 4000, 5000

RDMODE

Opcode	Instruction	Operation
ED 7F	RDMODE	CF = SU[0]

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 3000A/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	•					

Description

Sets the C flag to the value of bit 0 of SU. Bit 0 of SU is the current system/user mode.

RES *b*,*r*

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Resets bit *b* (any of the bits 0, 1, 2, 3, 4, 5, 6, or 7) of *r* (any of the registers A, B, C, D, E, H, or L). The bit is reset by performing a bitwise AND between the selected bit and its complement.

Bit Reset

`RES b,(HL)`

Opcode	Instruction	Operation
——	**RES *b*,(HL)**	**(HL) = (HL) & ~bit *b***
CB 86	RES 0,(HL)	(HL) = (HL) & ~bit 0
CB 8E	RES 1,(HL)	(HL) = (HL) & ~bit 1
CB 96	RES 2,(HL)	(HL) = (HL) & ~bit 2
CB 9E	RES 3,(HL)	(HL) = (HL) & ~bit 3
CB A6	RES 4,(HL)	(HL) = (HL) & ~bit 4
CB AE	RES 5,(HL)	(HL) = (HL) & ~bit 5
CB B6	RES 6,(HL)	(HL) = (HL) & ~bit 6
CB BE	RES 7,(HL)	(HL) = (HL) & ~bit 7

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	10	n/a	n/a
Rabbit 5000	10	10	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	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 in HL

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

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

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	13	n/a	n/a
Rabbit 5000	14	12	10

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	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:

- the sum of IX and the 8-bit signed displacement *d*, or
- the sum of IY and the 8-bit signed displacement *d*.

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

Return

2000, 3000, 4000, 5000

RET

Opcode	Instruction	Operation
C9	RET	PC_{low} = (SP) PC_{high} = (SP + 1) SP = SP + 2

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	8	n/a	n/a
Rabbit 5000	8	8	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Transfers execution from a subroutine to the program that called the subroutine by popping the return address from the stack into PC.

First, the low-order byte of PC is loaded with the data at the memory address in SP then the high-order byte of PC is loaded 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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	8	n/a	n/a
Rabbit 5000	8	8	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

If the condition *f* is false, the instruction is ignored. Otherwise, program execution continues at the address at the top of the stack. See “Condition Codes” on page 4 for a description of *f*.

RETI

Opcode	Instruction	Operation

ED 4D	RETI	IP = (SP)
		PC_{low} = (SP+1)
		PC_{high} = (SP+2)
		SP = SP+3

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	12	n/a	n/a
Rabbit 5000	12	12	10

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Loads IP with the data whose address is on the top of the stack, which should be the interrupt priority that was saved when the interrupt occurred. Then, loads PC from the stack, which should be the return address that was saved when the interrupt occurred. Next, the interrupt priority and the return address are popped off the stack by adding 3 to SP.

This is a chained-atomic instruction, meaning that an interrupt cannot take place between this instruction and the instruction following it.

Rotate Left Through Carry

RL BC
RL HL

Opcode	Instruction	Operation
62	RL BC	{CF,BC} = {BC,CF}
42	RL HL	{CF,HL} = {HL,CF}

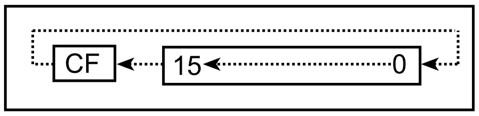
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

Description

Rotates BC or HL to the left with the C flag. 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.

Figure 1: Bit logic of the RL instruction



RL *bb*,BCDE

Opcode	Instruction	Operation

——	**RL *bb*,BCDE**	**{CF,BCDE} = {BCDE,CF}**
		***bb* = *bb* - 1**
		repeat while *bb* != 0

DD 68	RL 1,BCDE	{CF,BCDE} = {BCDE,CF}
		bb = *bb* - 1
		repeat while *bb* != 0

DD 69	RL 2,BCDE	{CF,BCDE} = {BCDE,CF}
		bb = *bb* - 1
		repeat while *bb* != 0

DD 6B	RL 4,BCDE	{CF,BCDE} = {BCDE,CF}
		bb = *bb* - 1
		repeat while *bb* != 0

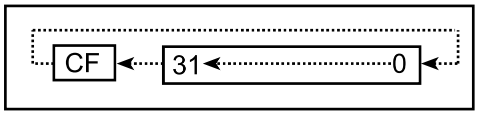
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

Description

Rotates BCDE to the left with the C flag. 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 31 moves to the C flag. This operation happens *b* number of times.

Figure 2: Bit logic of the RL instruction



RL *bb*,JKHL

Opcode	Instruction	Operation

——	**RL *bb*,JKHL**	**{CF,JKHL} = {JKHL,CF}**
		***bb* = *bb* - 1**
		repeat while *bb* != 0

FD 68	RL 1,JKHL	{CF,JKHL} = {JKHL,CF}
		bb = *bb* - 1
		repeat while *bb* != 0

FD 69	RL 2,JKHL	{CF,JKHL} = {JKHL,CF}
		bb = *bb* - 1
		repeat while *b* != 0

FD 6B	RL 4,JKHL	{CF,JKHL} = {JKHL,CF}
		bb = *bb* - 1
		repeat while *bb* != 0

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

Description

Rotates JKHL to the left with the C flag. 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 31 moves to the C flag. This operation happens *bb* number of times. See Figure 2 for an illustration.

RL DE

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

Description

Rotates DE to the left with the C flag. 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 1 for an illustration.

RL *r*

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

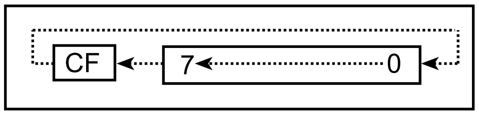
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

Description

Rotates *r* (any of the register A, B, C, D, E, H, or L) to the left with the C flag. 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 below.

Figure 3: The bit logic of the RL instruction.



RL (HL)

Opcode	Instruction	Operation
CB 16	RL (HL)	{CF,(HL)} = {(HL),CF}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	10	n/a	n/a
Rabbit 5000	10	10	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•			•	•

Description

Rotates to the left with the C flag the data whose address is HL.

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 3 for an illustration.

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.

Rotate Left Through Carry — 2000, 3000, 4000, 5000

`RL (IX+d)`
`RL (IY+d)`

Opcode	Instruction	Operation
DD CB *d* 16	RL (IX+*d*)	{CF,(IX + *d*)} = {(IX + *d*),CF}
FD CB *d* 16	RL (IY+*d*)	{CF,(IY + *d*)} = {(IY + *d*),CF}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	13	n/a	n/a
Rabbit 5000	14	12	10

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•			•	•

Description

Rotates to the left with the C flag the data whose address is:

- the sum of IX and the 8-bit signed displacement *d*, or
- the sum of IY and *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 3 for an illustration.

Example

If the sum of IX and *d* is 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 (IX+d)
```

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

Rotate Left A Through Carry — 2000, 3000, 4000, 5000

RLA

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	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 3 for an illustration.

Rotate Left Through A

RLB A,BCDE
RLB A,JKHL

Opcode	Instruction	Operation
DD 6F	RLB A,BCDE	{A,BCDE} = {BCDE,A}
FD 6F	RLB A,JKHL	{A,JKHL} = {JKHL,A}

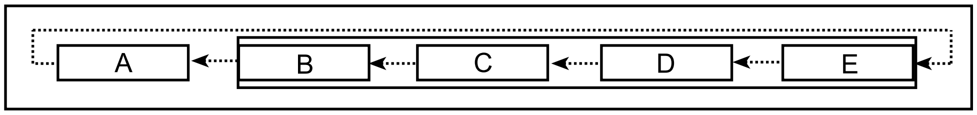
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Rotates BCDE or JKHL to the left with A. The bits are rotated 8 at a time, as illustrated in the figure below. Bits 0 through 23 are shifted 8 bits to bit positions 8 through 31. Bits 31 through 24 are shifted to A and A is shifted to bits 7 through 0.

Figure 4: The bit logic of the "RLB A, BCDE" instruction



Rotate Left Affect Carry

4000, 5000

RLC BC
RLC DE

Opcode	Instruction	Operation

60	RLC BC	BC = {BC[14,0],B[7]}
		CF = B[7]

50	RLC DE	DE = {DE[14,0],D[7]}
		CF = D[7]

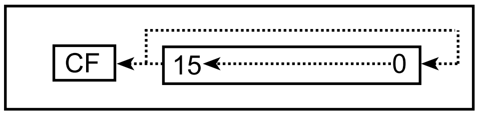
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

Description

Rotates BC or DE to the left (bit 0 moves to bit 1, etc.). The highest-order bit is rotated to the C flag and bit 0. See the figure below.

Figure 5: The bit logic of the RLC instruction



Rotate Left Affect Carry — 4000, 5000

`RLC bb,BCDE`

Opcode	Instruction	Operation

—	**RLC *bb*,BCDE**	**BCDE = {BCDE[30,0],B[7]}**
		CF = B[7]
		***bb* = *bb*-1**
		repeat while *bb* != 0

DD 68	RLC 1,BCDE	BCDE = {BCDE[30,0],B[7]}
		CF = B[7]
		bb = bb-1
		repeat while bb != 0

DD 69	RLC 2,BCDE	BCDE = {BCDE[30,0],B[7]}
		CF = B[7]
		bb = bb-1
		repeat while bb != 0

DD 6B	RLC 4,BCDE	BCDE = {BCDE[30,0],B[7]}
		CF = B[7]
		bb = bb-1
		repeat while bb != 0

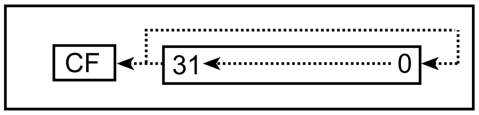
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

Description

Rotates BCDE to the left (bit 0 moves to bit 1, bit 1 moves to bit 2 etc.). Bit 31 moves to the C flag and bit 0. See the figure below.

Figure 6: The bit logic of the RLC operation.



This operation happens *bb* number of times.

Rotate Left Affect Carry 4000, 5000

`RLC` *bb*,`JKHL`

Opcode	Instruction	Operation

—	**RLC *bb*,JKHL**	**JKHL = {JKHL[30,0],J[7]}**
		 CF = J[7]
		 ***bb* = *bb*-1**
		 repeat while *bb* != 0

FD 68	RLC 1,JKHL	JKHL = {JKHL[30,0],J[7]}
		 CF = J[7]
		 bb = bb-1
		 repeat while bb != 0

FD 69	RLC 2,JKHL	JKHL = {JKHL[30,0],J[7]}
		 CF = J[7]
		 bb = bb-1
		 repeat while bb != 0

FD 6B	RLC 4,JKHL	JKHL = {JKHL[30,0],J[7]}
		 CF = J[7]
		 bb = bb-1
		 repeat while bb != 0

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

Description

Rotates JKHL to the left (bit 0 moves to bit 1, etc.) while bit 7 of the highest-order byte moves to bit 0 of the lowest-order byte and the C flag. This operation happens *bb* number of times. See Figure 6 for an illustration.

RLC *r*

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

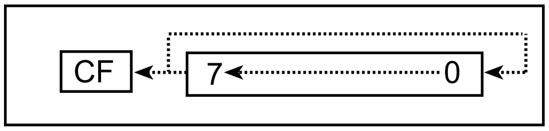
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

Description

Rotates *r* (any of the register A, B, C, D, E, H, or L) to the left. Each bit in the register moves to the next highest-order bit position (bit 0 moves to bit 1, etc.). Bit 7 moves to both bit 0 and the C flag.

Figure 7: The bit logic of the RLC instruction.



RLC (HL)

Opcode	Instruction	Operation

CB 06	RLC (HL)	(HL) = {(HL)[6,0],(HL)[7]}
		CF = (HL)[7]

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	10	n/a	n/a
Rabbit 5000	10	10	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•			•	•

Description

Rotates to the left the data whose address is HL.

Each bit moves to the next highest-order bit position (bit 0 moves to bit 1, etc.). Bit 7 moves to both bit 0 and the C flag. See Figure 7 for an illustration.

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.

Rotate Left Affect Carry

2000, 3000, 4000, 5000

`RLC (IX+d)`
`RLC (IY+d)`

Opcode	Instruction	Operation

DD CB *d* 06	RLC (IX+*d*)	(IX+*d*) = {(IX+*d*)[6,0],(IX+*d*)[7]}
		CF = (IX+*d*)[7]

FD CB *d* 06	RLC (IY+*d*)	(IY+*d*) = {(IY+*d*)[6,0],(IY+*d*)[7]}
		CF = (IY+*d*)[7]

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	13	n/a	n/a
Rabbit 5000	14	12	10

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•			•	•

Description

Rotates to the left the data whose address is:

- the sum of IX and the 8-bit signed displacement *d*, or
- the sum of IY and *d*.

Each bit moves to the next highest-order bit position (bit 0 moves to bit 1, etc.). Bit 7 moves to both bit 0 and the C flag. See Figure 7 for an illustration.

Example

If the sum of IX and *d* is 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 (IX+d)
```

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

Rotate Left

RLC 8,BCDE
RLC 8,JKHL

Opcode	Instruction	Operation
DD 4F	RLC 8,BCDE	BCDE = {CDE,B}
FD 4F	RLC 8,JKHL	JKHL = {KHL,J}

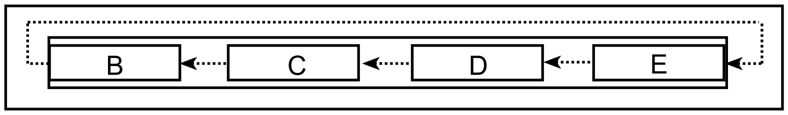
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Rotates BCDE or JKHL to the left. Each byte moves to the next highest-order byte position, with the highest-order byte moving to the lowest-order byte. See the figure below.

Figure 8: Bit logic of "RLC 8,BCDE" instruction



Rotate Left A Affect Carry

RLCA

Opcode	Instruction	Operation

07	RLCA	A = {A[6,0],A[7]}
		CF = A[7]

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	•	•	•			

Description

Rotates A to the left. 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 7 for an illustration.

Rotate Right Through Carry 4000, 5000

RR BC

Opcode	Instruction	Operation
63	RR BC	{BC,CF} = {CF,BC}

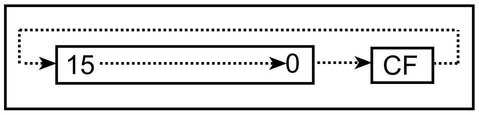
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

Description

Rotates to the right with the C flag the data in BC. See the figure below.

Figure 9: The bit logic of the RR instruction



Rotate Right Through Carry

RR *bb*,BCDE
RR *bb*,JKHL

Opcode	Instruction	Operation

—	**RR bb,BCDE**	**{BCDE,CF} = {CF,BCDE}**
		***bb* = *bb* - 1**
		repeat while *bb* != 0

DD 78	RR 1,BCDE	repeat the operation 1 time
DD 79	RR 2,BCDE	repeat the operation 2 times
DD 7B	RR 4,BCDE	repeat the operation 4 times

—	**RR *bb*,JKHL**	**{JKHL,CF} = {CF,JKHL}**
		***bb* = *bb* - 1**
		repeat while *bb* != 0

FD 78	RR 1,JKHL	repeat the operation 1 time
FD 79	RR 2,JKHL	repeat the operation 2 times
FD 7B	RR 4,JKHL	repeat the operation 4 times

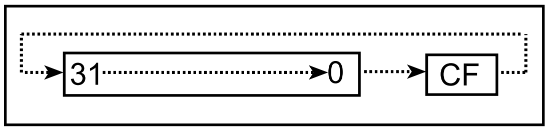
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

Description

Rotates to the right with the C flag the data in BCDE or JKHL.

Figure 10: The bit logic of the RR instruction.



This operation happens *bb* times.

Rotate Right Through Carry

2000, 3000, 4000, 5000

RR DE

RR HL

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

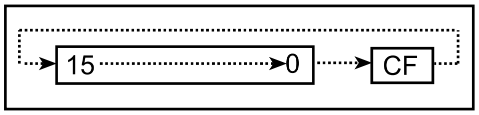
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

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 11 below for an illustration.

Figure 11: The bit logic for RR instruction.



Rotate Right Through Carry

RR IX
RR IY

Opcode	Instruction	Operation
DD FC	RR IX	{IX,CF} = {CF,IX}
FD FC	RR IY	{IY,CF} = {CF,IY}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•				

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 9 for an illustration.

Rotate Right Through Carry 2000, 3000, 4000, 5000

`RR r`

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

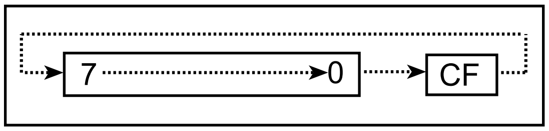
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

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 the figure below.

Figure 12: The bit logic for the RR instruction.



`RR (HL)`

Opcode	Instruction	Operation
CB 1E	RR (HL)	{(HL),CF} = {CF,(HL)}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	10	n/a	n/a
Rabbit 5000	10	10	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•			•	•

Description

Rotates to the right with the C flag the data whose address is HL.

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 12 for an illustration.

Rotate Right Through Carry 2000, 3000, 4000, 5000

`RR (IX+d)`
`RR (IY+d)`

Opcode	Instruction	Operation
DD CB *d* 1E	RR (IX+*d*)	{(IX+*d*),CF} = {CF,(IX+*d*)}
FD CB *d* 1E	RR (IY+*d*)	{(IY+*d*),CF} = {CF,(IY+*d*)}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	13	n/a	n/a
Rabbit 5000	14	12	10

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•			•	•

Description

Rotates to the right with the C flag the data whose address is:

- the sum of IX and the 8-bit signed displacement *d*, or
- the sum of IY and the 8-bit signed 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 12 for an illustration.

Rotate Right A Through Carry

2000, 3000, 4000, 5000

`RRA`

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	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 12 for an illustration.

Rotate Right Through A 4000, 5000

`RRB A,BCDE`
`RRB A,JKHL`

Opcode	Instruction	Operation
DD 7F	RRB A,BCDE	{A, BCDE} = {E, A, BCD}
FD 7F	RRB A,JKHL	{A, JKHL} = {L, A, JKH}

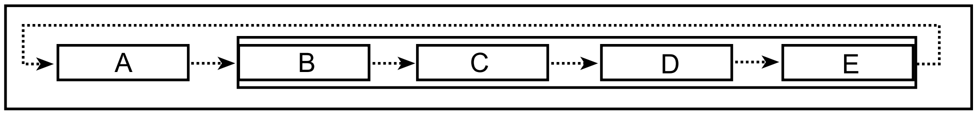
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Rotates 8 bits to the right with A the data in BCDE or JKHL. For example, the data in B moves to C, while the data in C moves to D. The low-order byte moves to A and A moves to the high-order byte. See the figure below.

Figure 13: The bit logic of the “RRA 8,BCDE” instruction.



Rotate Right Affect Carry

RRC BC
RRC DE

Opcode	Instruction	Operation

61	RRC BC	BC = {B[0],BC[15,1]}
		CF = C[0]

51	RRC DE	DE = {D[0],DE[15,1]}
		CF = E[0]

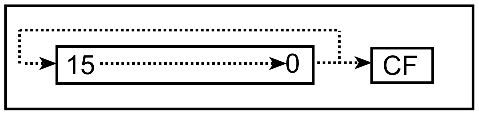
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

Description

Rotates to the right the C flag with the data in BC or DE. Each bit in the register moves to the next lowest-order bit position (bit 15 moves to bit 14, etc.) while bit 0 moves to both bit 15 and the C flag. See the figure below.

Figure 14: The bit logic of RRC.



Rotate Right Affect Carry

RRC *bb*,BCDE
RRC *bb*,JKHL

Opcode	Instruction	Operation

—	**RRC *bb*,BCDE**	**BCDE = {B[7],BCDE[31,1]}**
		CF = E[0]
		***bb* = *bb* - 1**
		repeat while *bb* != 0

DD 58	RRC 1,BCDE	repeat the operation 1 time
DD 59	RRC 2,BCDE	repeat the operation 2 times
DD 5B	RRC 4,BCDE	repeat the operation 4 times

—	**RRC b,JKHL**	**JKHL = {J[7],JKHL[31,1]}**
		CF = L[0]
		***bb* = *bb* - 1**
		repeat while *bb* !=0

FD 58	RRC 1,JKHL	repeat the operation 1 time
FD 59	RRC 2,JKHL	repeat the operation 2 times
FD 5B	RRC 4,JKHL	repeat the operation 4 times

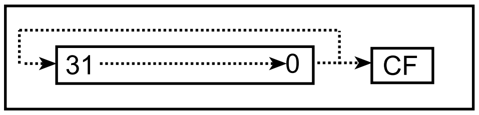
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

Description

Rotates to the right the data in BCDE or JKHL. Each bit in the register moves to the next lowest-order bit position (bit 31 moves to bit 30, etc.) while bit 0 moves to both bit 31 and the C flag.

Figure 15: The bit logic of RRC.



This operation happens *bb* number of times.

RRC *r*

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

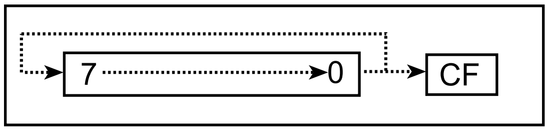
Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

Description

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

Each bit 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.

Figure 16: The bit logic of the RRC instruction.



RRC (HL)

Opcode	Instruction	Operation

CB 0E	RRC (HL)	(HL) = {(HL)[0],(HL)[7,1]}
		CF = (HL)[0]

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	10	n/a	n/a
Rabbit 5000	10	10	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•			•	•

Description

Rotates to the right the data whose address is HL.

Each bit 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 16 for an illustration.

Rotate Right Affect Carry

`RRC (IX+d)`
`RRC (IY+d)`

Opcode	Instruction	Operation

DD CB *d* 0E	RRC (IX+*d*)	(IX + *d*) = {(IX + *d*)[0], (IX + *d*)[7,1]}
		CF = (IX + *d*)[0]

FD CB *d* 0E	RRC (IY+*d*)	(IY + *d*) = {(IY + *d*)[0], (IY + *d*)[7,1]}
		CF = (IY + *d*)[0]

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	13	n/a	n/a
Rabbit 5000	14	12	10

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•			•	•

Description

Rotates to the right the data whose address is:

- the sum of IX and the 8-bit signed displacement *d*, or
- the sum of IY and the 8-bit signed displacement *d*.

Each bit 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 16 for an illustration.

Rotate Right

```
RRC 8,BCDE
RRC 8,JKHL
```

Opcode	Instruction	Operation
DD 5F	RRC 8,BCDE	BCDE = {E, BCD}
FD 5F	RRC 8,JKHL	JKHL = {L, JKH}

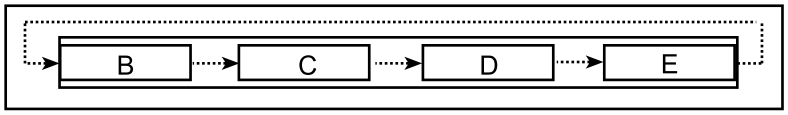
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Rotates BCDE or JKHL to the right. Each byte in the register moves to the next lowest-order byte position. E.g., the byte in B is loaded into C; the byte that was in C is loaded into D; the byte that was in D is loaded into E; and the byte that was in E is loaded into B.

Figure 17: The bit logic of the "RRC 8,BCDE" instruction



RRCA

Opcode	Instruction	Operation

0F	RRCA	A = {A[0],A[7,1]}
		CF = A[0]

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	•	•	•			

Description

Rotates to the right the data in A. Each bit 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 16 for an illustration.

RST *v*

Opcode	Instruction	Operation
——	**RST *v***	**(SP-1) = PC_{high}** **(SP-2) = PC_{low}** **SP = SP-2;** **PC = Restart Address**

D7	RST 10	(SP-1) = PC_{high}; (SP-2) = PC_{low}; SP = SP-2;
		PC = IIR:0x20

DF	RST 18	(SP-1) = PC_{high}; (SP-2) = PC_{low}; SP = SP-2;
		PC = IIR:0x30

E7	RST 20	(SP-1) = PC_{high}; (SP-2) = PC_{low}; SP = SP-2;
		PC = IIR:0x40

EF	RST 28	(SP-1) = PC_{high}; (SP-2) = PC_{low}; SP = SP-2;
		PC = IIR:0x50

FF	RST 38	(SP-1) = PC_{high}; (SP-2) = PC_{low}; SP = SP-2;
		PC = IIR:0x70

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	8	n/a	n/a
Rabbit 5000	11	11	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Transfers program execution to the interrupt vector address specified by IIR:*v*, where IIR is the address of the interrupt vector table and *v* is the offset. The vector table is always on a 100h boundary. Its address 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.

Subtract Through Carry

`SBC A,n`

Opcode	Instruction	Operation
DE *n*	SBC A,*n*	A = A - *n* - CF

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•			

Description

Subtracts the C flag and the 8-bit constant *n* from 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 unchanged if A is equal to the data being subtracted from it.

The Rabbit 4000/5000 assemblers view “SBC A,n” and “SBC n” as equivalent instructions. In the latter case, A is used even though it is not explicitly stated.

`SBC A,r`

Opcode	Instruction	Operation
——	**SBC A,*r***	**A = A - *r* - CF**
9F	SBC A,A	A = A - A - CF
98	SBC A,B	A = A - B - CF
99	SBC A,C	A = A - C - CF
9A	SBC A,D	A = A - D - CF
9B	SBC A,E	A = A - E - CF
9C	SBC A,H	A = A - H - CF
9D	SBC A,L	A = A - L - CF

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000	2	n/a	n/a

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•			

Description

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 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 unchanged if A is equal to the data being subtracted from it.

Subtract Through Carry

`SBC A,r`

Opcode	Instruction	Operation
—	**SBC A,*r***	**A = A - *r* - CF**
7F 9F	SBC A,A	A = A - A - CF
7F 98	SBC A,B	A = A - B - CF
7F 99	SBC A,C	A = A - C - CF
7F 9A	SBC A,D	A = A - D - CF
7F 9B	SBC A,E	A = A - E - CF
7F 9C	SBC A,H	A = A - H - CF
7F 9D	SBC A,L	A = A - L - CF

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•			

Description

Subtracts the C flag and the data in *r* (one of A, B, C, D, E, H or L) from A. The result is stored in A.

The Rabbit 4000/5000 assemblers view "SBC A,r" and "SBC r" as equivalent instructions. In the latter case, A is used even though it is not explicitly stated.

The opcodes for these instructions are different than the same instructions in the Rabbit 2000, 3000 and 3000A.

`SBC A,(HL)`

Opcode	Instruction	Operation
9E	SBC A,(HL)	A = A - (HL) - CF

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000	5	n/a	n/a

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•		•	

Description

Subtracts the C flag and the data whose address is the data in HL 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 unchanged if A is equal to the data being subtracted from it.

SBC A,(HL)

Opcode	Instruction	Operation
7F 9E	SBC A,(HL)	A = A - (HL) - CF

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	7	n/a	n/a
Rabbit 5000	7	7	5

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•		•	

Description

Subtracts the C flag and the data whose address is in HL from 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 unchanged if A is equal to the data being subtracted from it.

The Rabbit 4000/5000 assemblers view “SBC A,(HL)” and “SBC (HL)” as equivalent instructions. In the latter case, A is used even though it is not explicitly stated.

The opcode for this instruction is different than the same instruction in the Rabbit 2000, 3000 and 3000A.

SBC HL,*ss*

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•			

Description

Subtracts the C flag and the data in *ss* (any of BC, DE, HL, or SP) from HL. The result is stored in HL. These operations output an inverted carry:

• The C flag is set if HL is less than the data being subtracted from it.

• The C flag is cleared if HL is greater than the data being subtracted from it.

• The C flag is unchanged if HL is equal to the data being subtracted from it.

`SBC (IX+d)`
`SBC (IY+d)`

Opcode	Instruction	Operation
DD 9E *d*	SBC (IX+*d*)	A = A - (IX+*d*) - CF
FD 9E *d*	SBC (IY+*d*)	A = A - (IY+*d*) - CF

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	9	n/a	n/a
Rabbit 5000	10	9	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•		•	

Description

Subtracts the C flag and the data whose address is:

- the sum of IX and the 8-bit signed displacement *d*, or
- the sum of IY and the 8-bit signed displacement *d*

from 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 unchanged if A is equal to the data being subtracted from it.

The Rabbit 4000/5000 assemblers view “SBC A,(IX+d)” and “SBC (IX+d)” as equivalent instructions. The same is true for “SBC A,(IY+d)” and “SBC (IY+d).”

SBOX 4000, 5000

SBOX A

Opcode	Instruction	Operation
ED 02	SBOX A	A = sbox(A)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-		•			

Description

Sbox is a 256-byte lookup table used by the AES-128 cipher. A contains the index into the table and is replaced by the value at that index location.

SCF

Opcode	Instruction	Operation
37	SCF	CF = 1

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	1	•				

Description

Sets the C flag.

Set Bit

2000, 3000, 4000, 5000

`SET b,r`

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	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).

Example

If A contains 1100 0000, after the execution of the operation:

```
SET 3,A
```

A contains 1100 1000.

SET *b*,(HL)

Opcode	Instruction	Operation
—	**SET *b*,(HL)**	**(HL) = (HL) | *b***
CB C6	SET 0,(HL)	(HL) = (HL) | bit 0
CB CE	SET 1,(HL)	(HL) = (HL) | bit 1
CB D6	SET 2,(HL)	(HL) = (HL) | bit 2
CB DE	SET 3,(HL)	(HL) = (HL) | bit 3
CB E6	SET 4,(HL)	(HL) = (HL) | bit 4
CB EE	SET 5,(HL)	(HL) = (HL) | bit 5
CB F6	SET 6,(HL)	(HL) = (HL) | bit 6
CB FE	SET 7,(HL)	(HL) = (HL) | bit 7

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	10	n/a	n/a
Rabbit 5000	10	10	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	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.

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

Opcode	Instruction	Operation
—	**SET *b*,(IX+*d*)**	**(IX+*d*) = (IX+*d*) | *b***
DD CB *d* C6	SET 0,(IX+*d*)	(IX+*d*) = (IX+*d*) | bit 0
DD CB *d* CE	SET 1,(IX+*d*)	(IX+*d*) = (IX+*d*) | bit 1
DD CB *d* D6	SET 2,(IX+*d*)	(IX+*d*) = (IX+*d*) | bit 2
DD CB *d* DE	SET 3,(IX+*d*)	(IX+*d*) = (IX+*d*) | bit 3
DD CB *d* E6	SET 4,(IX+*d*)	(IX+*d*) = (IX+*d*) | bit 4
DD CB *d* EE	SET 5,(IX+*d*)	(IX+*d*) = (IX+*d*) | bit 5
DD CB *d* F6	SET 6,(IX+*d*)	(IX+*d*) = (IX+*d*) | bit 6
DD CB *d* FE	SET 7,(IX+*d*)	(IX+*d*) = (IX+*d*) | bit 7
—	**SET *b*,(IY+*d*)**	**(IY+*d*) = (IY+*d*) | *b***
FD CB *d* C6	SET 0,(IY+*d*)	(IY+*d*) = (IY+*d*) | bit 0
FD CB *d* CE	SET 1,(IY+*d*)	(IY+*d*) = (IY+*d*) | bit 1
FD CB *d* D6	SET 2,(IY+*d*)	(IY+*d*) = (IY+*d*) | bit 2
FD CB *d* DE	SET 3,(IY+*d*)	(IY+*d*) = (IY+*d*) | bit 3
FD CB *d* E6	SET 4,(IY+*d*)	(IY+*d*) = (IY+*d*) | bit 4
FD CB *d* EE	SET 5,(IY+*d*)	(IY+*d*) = (IY+*d*) | bit 5
FD CB *d* F6	SET 6,(IY+*d*)	(IY+*d*) = (IY+*d*) | bit 6
FD CB *d* FE	SET 7,(IY+*d*)	(IY+*d*) = (IY+*d*) | bit 7

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	13	n/a	n/a
Rabbit 5000	12	12	10

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	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 sum of the data in IX and a displacement *d*, or
- the sum of the data in IY and a displacement *d*.

Return From User Interrupt 4000, 5000

`SETSYSP mn`

Opcode	Instruction	Operation

ED B1 *n* *m*	SETSYSP *mn*	SU={SU[1:0],SU[7:2]}
		tmp_{low} = (SP)
		tmp_{high} = (SP + 1)
		SP = SP + 2
		if {tmp != mn}
		  System Violation

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	12	n/a	n/a
Rabbit 5000	12	10	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

This instruction is used to handle user mode interrupts in system/user mode. It sets the current processor mode to the previous processor mode by rotating two places to the right the bits of SU. Bits 1 and 0 are moved to bit positions 7 and 6 respectively. 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.

This is a chained-atomic instruction, meaning that an interrupt cannot take place between this instruction and the instruction following it.

Set User Mode 3000A, 4000, 5000

SETUSR

Opcode	Instruction	Operation
ED 6F	SETUSR	SU={SU[5:0],0x01}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 3000A/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	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 is a chained-atomic instruction, meaning that an interrupt cannot take place between this instruction and the instruction following it.

Enter User Interrupt 4000, 5000

`SETUSRP` *`mn`*

Opcode	Instruction	Operation

ED B5 *n m*	SETUSRP *mn*	SU = {SU[7:2],01}
		(SP - 1) = *m*
		(SP - 2) = *n*
		SP = SP - 2

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	15	n/a	n/a
Rabbit 5000	16	14	12

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Sets the current processor mode by setting SU bit 1 to zero and bit 0 to one.

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.

This instruction is used to handle user mode interrupts in system/user mode.

This is a chained-atomic instruction, meaning that an interrupt cannot take place between this instruction and the instruction following it.

Smart Jump 4000, 5000

SJP label

Description

This pseudo instruction resolves to one of the following:

- 8-bit relative jump (see `JR label`)
- 16-bit absolute jump (see `JP mn`)
- 20-bit absolute jump (see `LJP x,mn`)

If “label” has not yet been resolved, the assembler inserts nops into the code to allow for the longest possible jump instruction.

For compatibility with future revisions of the compiler, the “sjp” instruction should not be used if the number of bytes in the binary code must stay constant across compilers.

Shift Left Arithmetic

SLA *bb*,BCDE
SLA *bb*,JKHL

Opcode	Instruction	Operation

——	**SLA *bb*,BCDE**	**BCDE = {BCDE[30,0],0}**
		CF = B[7]
		***bb* = *bb* - 1**
		repeat while *bb*!=0

DD 88	SLA 1,BCDE	repeat the operation 1 time
DD 89	SLA 2,BCDE	repeat the operation 2 times
DD 8B	SLA 4,BCDE	repeat the operation 4 times

——	**SLA *b*,JKHL**	**JKHL = {JKHL[30,0],0}**
		CF = J[7]
		bb = bb - 1
		repeat while bb!=0

FD 88	SLA 1,JKHL	repeat the operation 1 time
FD 89	SLA 2,JKHL	repeat the operation 2 times
FD 8B	SLA 4,JKHL	repeat the operation 4 times

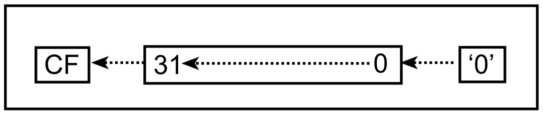
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

Description

Arithmetically shifts to the left the bits of BCDE or JKHL. Bits 0 through 30 are each shifted to the next highest-order bit position (bit 0 moves to bit 1, etc.). Bit 31 is shifted to the C flag. Bit 0 is reset.

Figure 18: The bit logic of the SLA instruction.



The operation happens *bb* number of times, which can be 1, 2 or 4.

Shift Left Arithmetic 2000, 3000, 4000, 5000

`SLA r`

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

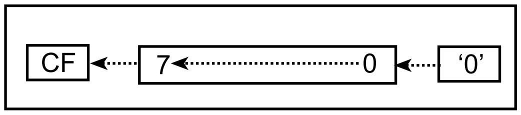
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

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 the figure below.

Figure 19: The bit logic of the SLA instruction.



`SLA (HL)`

Opcode	Instruction	Operation

CB 26	SLA (HL)	(HL) = {(HL)[6,0],0}
		CF = (HL)[7]

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	10	n/a	n/a
Rabbit 5000	10	10	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•			•	•

Description

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

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 19 for an illustration.

Shift Left Arithmetic 2000, 3000, 4000, 5000

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

Opcode	Instruction	Operation

DD CB *d* 26	SLA (IX+*d*)	(IX + *d*) = {(IX + *d*)[6,0],0}
		CF = (IX+*d*)[7]

FD CB *d* 26	SLA (IY+*d*)	(IY + *d*) = {(IY + *d*)[6,0],0}
		CF = (IY+*d*)[7]

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	13	n/a	n/a
Rabbit 5000	14	12	10

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•			•	•

Description

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

- the sum of IX and the 8-bit signed displacement *d*, or
- the sum of IY and the 8-bit signed 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 19 for an illustration.

Shift Left Logical

SLL *bb*,BCDE
SLL *bb*,JKHL

Opcode	Instruction	Operation

——	**SLL *bb*,BCDE**	**BCDE = {BCDE[30,0],0}**
		CF = B[7]
		***bb* = *bb* - 1**
		repeat while *bb* != 0

DD A8	SLL 1,BCDE	the operation happens 1 time
DD A9	SLL 2,BCDE	the operation happens 2 times
DD AB	SLL 4,BCDE	the operation happens 4 times

——	**SLL *bb*,JKHL**	**JKHL = {JKHL[30,0],0}**
		CF = J[7]
		bb = bb - 1
		repeat while bb != 0

FD A8	SLL 1,JKHL	the operation happens 1 time
FD A9	SLL 2,JKHL	the operation happens 2 times
FD AB	SLL 4,JKHL	the operation happens 4 times

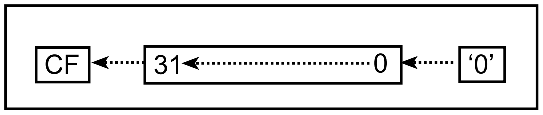
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

Description

Shifts to the left the bits of the data in register BCDE or JKHL. Bits 0 through 30 are each shifted to the next highest-order bit position (bit 0 moves to bit 1, etc.). The highest-order bit (bit 31 of BCDE or JKHL) is shifted to the C flag. Bit 0 is reset. See the figure below.

Figure 20: The bit logic of the SLL instruction



The operation happens *bb* number of times, which can be 1, 2 or 4.

Shift Right Arithmetic

4000, 5000

SRA *bb*,BCDE
SRA *bb*,JKHL

Opcode	Instruction	Operation

——	**SRA *bb*,BCDE**	**BCDE = {B[7],BCDE[31,1]}**
		CF = E[0]
		bb = bb - 1
		repeat while bb != 0

DD 98	SRA 1,BCDE	repeat the operation 1 time
DD 99	SRA 2,BCDE	repeat the operation 2 times
DD 9B	SRA 4,BCDE	repeat the operation 4 times

——	**SRA *bb*,JKHL**	**JKHL = {J[7],JKHL[31,1]}**
		CF = L[0]
		bb = bb - 1
		repeat while bb != 0

FD 98	SRA 1,JKHL	repeat the operation 1 time
FD 99	SRA 2,JKHL	repeat the operation 2 times
FD 9B	SRA 4,JKHL	repeat the operation 4 times

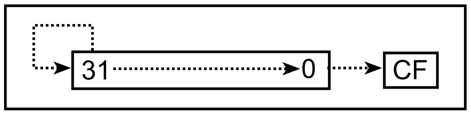
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

Description

Arithmetically shifts to the right the bits of the data in register BCDE or JKHL. Bits 1 through 31 are each shifted to the next lowest-order bit position. The highest-order bit of BCDE or JKHL is shifted into itself and the lowest-order bit is shifted to the C flag. See the figure below.

Figure 21: The bit logic of the SRA instruction.



The instruction repeats the number of times specified by `bb`, which can be 1, 2 or 4.

SRA *r*

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

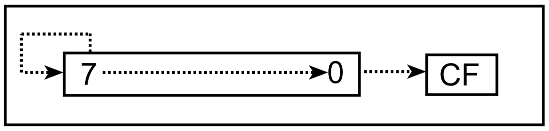
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

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 the figure below.

Figure 22: The bit logic of the SRA instruction.



`SRA (HL)`

Opcode	Instruction	Operation

CB 2E	SRA (HL)	(HL) = {(HL)[7],(HL)[7,1]}
		CF = (HL)[0]

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	10	n/a	n/a
Rabbit 5000	10	10	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•			•	•

Description

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

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 22 for an illustration.

`SRA (IX+d)`
`SRA (IY+d)`

Opcode	Instruction	Operation

DD CB *d* 2E	SRA (IX+*d*)	(IX+*d*) = {(IX+*d*)[7],(IX+*d*)[7,1]}
		CF = (IX+*d*)[0]

FD CB *d* 2E	SRA (IY+*d*)	(IY+*d*) = {(IY+*d*)[7],(IY+*d*)[7,1]}
		CF = (IY+*d*)[0]

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	13	n/a	n/a
Rabbit 5000	14	12	10

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•			•	•

Description

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

- the sum of IX and the 8-bit signed displacement *d*, or
- the sum of IY and the 8-bit signed 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 22 for an illustration.

Shift Right Logical

SRL *bb*,BCDE
SRL *bb*,JKHL

Opcode	Instruction	Operation

——	**SRL *bb*,BCDE**	**BCDE = {0,BCDE[31,1]}**
		CF = E[0]
		bb = bb - 1
		repeat while bb != 0

DD B8	SRL 1,BCDE	repeat the operation 1 time
DD B9	SRL 2,BCDE	repeat the operation 2 times
DD BB	SRL 4,BCDE	repeat the operation 4 times

——	**SRL *bb*,JKHL**	**JKHL = {0,JKHL[31,1]}**
		CF = L[0]
		bb= bb - 1
		repeat while bb != 0

FD B8	SRL 1,JKHL	repeat the operation 1 time
FD B9	SRL 2,JKHL	repeat the operation 2 times
FD BB	SRL 4,JKHL	repeat the operation 4 times

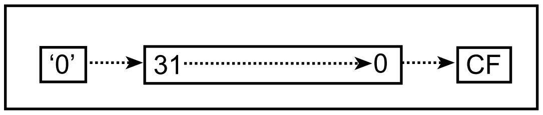
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

Description

Shifts to the right the bits of the data in register BCDE or JKHL. Bits 1 through 31 are each shifted to the next lowest-order bit position (bit 31 moves to bit 30, etc.). The lowest-order bit (bit 0 of E or L) is shifted to the C flag.

Figure 23: The bit logic of the SRL instruction.



The instruction repeats *bb* number of times, which can be 1, 2 or 4.

`SRL` *`r`*

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

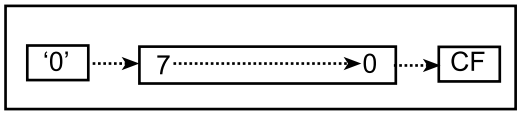
Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•	•			

Description

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 shifts to the C flag. Bit 7 is reset. See the figure below.

Figure 24: The bit logic of the SRL instruction.



SRL (HL)

Opcode	Instruction	Operation

CB 3E	SRL (HL)	(HL) = {0,(HL)[7,1]}
		CF = (HL)[0]

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	10	n/a	n/a
Rabbit 5000	10	10	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•			•	•

Description

Shifts to the right the bits of the data whose address is HL.

Each bit is shifted to the next lowest-order bit position (Bit 7 shifts to bit 6, etc.) Bit 0 shifts to the C flag. Bit 7 is reset. See Figure 24 for an illustration.

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

Opcode	Instruction	Operation

DD CB *d* 3E	SRL (IX+*d*)	(IX + *d*) = {0,(IX + *d*)[7,1]}
		CF = (IX + *d*)[0]

FD CB *d* 3E	SRL (IY+*d*)	(IY + *d*) = {0,(IY + *d*)[7,1]}
		CF = (IY + *d*)[0]

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	13	n/a	n/a
Rabbit 5000	14	12	10

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	•	•			•	•

Description

Shifts to the right the bits of the data whose address is

- the sum of IX and the 8-bit signed displacement *d*, or
- the sum of IY and the 8-bit signed displacement *d*.

Each bit is shifted to the next lowest-order bit position (Bit 7 shifts to bit 6, etc.) Bit 0 shifts to the C flag. Bit 7 is reset. See Figure 24 for an illustration.

SUB HL,DE
SUB HL,JK

Opcode	Instruction	Operation
55	SUB HL,DE	HL = HL - DE
45	SUB HL,JK	HL = HL - JK

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	•	•	•			

Description

Subtracts from the data in HL the data in DE or JK. The result is stored in HL.

SUB JKHL,BCDE

Opcode	Instruction	Operation
ED D6	SUB JKHL,BCDE	JKHL = JKHL - BCDE

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	•	•	•			

Description

Subtracts from the data in JKHL the data in BCDE. The result is stored in JKHL.

SUB *n*

Opcode	Instruction	Operation
D6 *n*	SUB *n*	A = A - *n*

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•			

Description

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

The Rabbit 4000/5000 assemblers view “SUB A,n” and “SUB n” as equivalent instructions.

SUB *r*

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000	2	n/a	n/a

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•			

Description

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

SUB *r*

Opcode	Instruction	Operation
——	**SUB *r***	**A = A - *r***
7F 97	SUB A	A = A - A
7F 90	SUB B	A = A - B
7F 91	SUB C	A = A - C
7F 92	SUB D	A = A - D
7F 93	SUB E	A = A - E
7F 94	SUB H	A = A - H
7F 95	SUB L	A = A - L

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•			

Description

Subtracts *r* (any of the registers A, B, C, D, E, H, or L) from A. The result is stored in A. The Rabbit 4000/5000 assemblers view "SUB A,r" and "SUB r" as equivalent instructions.

The opcodes for these instructions are different than the same instructions in the Rabbit 2000, 3000 and 3000A.

SUB (HL)

Opcode	Instruction	Operation
96	SUB (HL)	A = A - (HL)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000	5	n/a	n/a

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•		•	

Description

Subtracts the data whose address is in HL from A.The result is stored in A.

SUB (HL)

Opcode	Instruction	Operation
7F 96	SUB (HL)	A = A - (HL)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	7	n/a	n/a
Rabbit 5000	5	5	5

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•		•	

Description

Subtracts the data whose address is in HL from A.The result is stored in A.

The Rabbit 4000/5000 assemblers view “SUB A,(HL)” and “SUB (HL)” as equivalent instructions.

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

Opcode	Instruction	Operation
DD 96 *d*	SUB (IX+*d*)	A = A - (IX + *d*)
FD 96 *d*	SUB (IY+*d*)	A = A - (IY + *d*)

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	9	n/a	n/a
Rabbit 5000	10	9	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	V	•	•	•		•	

Description

Subtracts the data whose address is:

- the sum of IX and the 8-bit signed displacement *d*, or
- the sum of IY and *d*.

from A. The result is stored in A.

The Rabbit 4000/5000 assemblers view “SUB A,(IX+d)” and “SUB (IX+d)” as equivalent instructions.The same is true for “SUB A,(IY+d)” and “SUB (IY+d).”

System/User Restore 3000A, 4000, 5000

SURES

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 3000A/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

“Pops” the current mode off the SU register, returning the processor mode to the previous mode by rotating SU two bits to the right.

This is a chained-atomic instruction, meaning that an interrupt cannot take place between this instruction and the instruction following it.

System Call 3000A, 4000, 5000

SYSCALL

Opcode	Instruction	Operation
ED 75	SYSCALL	SP = SP-2; PC = {R,0x60}

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	10	n/a	n/a
Rabbit 5000	13	13	11

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	-					

Description

Pushes PC on 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.

SYSCALL is essentially a new RST opcode, added to allow access to system space without using one of the existing RST opcodes. It will put the processor into System mode and execute code in the corresponding interrupt-vector table entry.

System Return 4000, 5000

SYSRET

Opcode	Instruction	Operation

ED 83	SYSRET	SU = (SP)
		PC_{low} = (SP + 1)
		PC_{high} = (SP + 2)
		SP = SP + 3

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	12	n/a	n/a
Rabbit 5000	12	12	10

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
–	–	–	–					

Description

Return and restore SU stack.

Test for Zero

TEST HL

Opcode	Instruction	Operation
4C	TEST HL	HL | 0

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	2	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•				

Description

Test HL for zero by performing a bitwise OR of HL and zero.

Test for Zero 4000, 5000

TEST

Opcode	Instruction	Operation
ED 4C	TEST BC	BC | 0
DD 5C	TEST BCDE	BCDE | 0
DD 4C	TEST IX	IX | 0
FD 4C	TEST IY	IY | 0
FD 5C	TEST JKHL	JKHL | 0

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•				

Description

These instructions test registers for zero by performing a bitwise OR of the register and zero.

UMA

Opcode	Instruction	Operation
ED C0	UMA	{CF:DE’:(HL)} = (IX) + [(IY)* DE + DE’ + CF] BC = BC - 1; IX = IX + 1 IY = IY + 1; HL = HL + 1 repeat while BC != 0

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 3000A/4000	8+8i	n/a	n/a
Rabbit 5000	8+8i	8+8i	6+8i

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	•					

Description

Performs the following operation:

`{CF:DE':(HL)} = (IX) + [(IY) * DE + DE' + CF];`

where HL, IX, and IY increment after each byte, repeated BC times. This fundamental operation allows the addition or subtraction of two arbitrarily-long unsigned integers after one is scaled by a single-byte value. This operation is common in many cryptographic operations.

The above operation results in a 24-bit value. The lowest 8 bits of this value are stored in memory at the address in HL, and the upper 16 bits are stored in the alternate register DE’.

Interrupts can occur between different repeats, but not within an iteration.

Unsigned Multiplication and Subtraction 3000A, 4000, 5000

UMS

Opcode	Instruction	Operation

ED C8	UMS	{CF:DE':(HL)} = (IX) - [(IY)*DE+DE'+CF]
		BC = BC - 1; IX = IX + 1
		IY = IY + 1; HL = HL + 1
		repeat while BC != 0

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 3000A/4000	8+8i	n/a	n/a
Rabbit 5000	8+8i	8+8i	6+8i

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
-	-	-	•					

Description

Performs the following operation:

```
{CF:DE':(HL)} = (IX) - [(IY) * DE + DE' + CF];
```

where HL, IX, and IY increment after each byte, repeated BC times. This fundamental operation allows the addition or subtraction of two arbitrarily-long unsigned integers after one is scaled by a single-byte value. This operation is common in many cryptographic operations.

The above operation results in a 24-bit value. The lowest 8 bits of this value are stored in memory at the address in HL, and the upper 16 bits are stored in the alternate register DE'.

Interrupts can occur between different repeats, but not within an iteration.

XOR HL,DE

Opcode	Instruction	Operation
54	XOR HL,DE	HL = HL ^ DE

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•			

Description

This instruction performs an exclusive OR operation between the word in HL and the word in DE. The result is stored in HL.

Exclusive OR 4000, 5000

XOR JKHL,BCDE

Opcode	Instruction	Operation
ED EE	XOR JKHL,BCDE	JKHL = JKHL ^ BCDE

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•			

Description

This instruction performs an exclusive OR operation between the 32-bit value in JKHL and the 32-bit value in BCDE. The result is stored in JKHL.

XOR *n*

Opcode	Instruction	Operation
EE *n*	XOR *n*	A = [A & ~*n*] | [~A & *n*]

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	4	n/a	n/a
Rabbit 5000	4	4	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•			

Description

Performs an exclusive OR operation between the byte in A and the 8-bit constant *n*. The result is stored in A.

The Rabbit 4000/5000 assemblers view “XOR A,n” and “XOR n” as equivalent instructions.

XOR *r*

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

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000	2	n/a	n/a

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•			

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 result is stored in A.

XOR *r*

Opcode	Instruction	Operation
——	**XOR *r***	**A = [A & ~*r*] | [~A & *r*]**
AF	XOR A	A = 0
7F A8	XOR B	A = [A & ~B] | [~A & B]
7F A9	XOR C	A = [A & ~C] | [~A & C]
7F AA	XOR D	A = [A & ~D] | [~A & D]
7F AB	XOR E	A = [A & ~E] | [~A & E]
7F AC	XOR H	A = [A & ~H] | [~A & H]
7F AD	XOR L	A = [A & ~L] | [~A & L]

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	4	n/a	n/a
Rabbit 5000	2	2	2

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•			

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 result is stored in A.

The Rabbit 4000/5000 assemblers view “XOR A,r” and “XOR r” as equivalent instructions.

XOR (HL)

Opcode	Instruction	Operation
AE	XOR (HL)	A = [A & ~(HL)] | [~A & (HL)]

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000	5	n/a	n/a

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•		•	

Description

Performs an exclusive OR operation between A and the data whose address is in HL. 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 (HL)

Opcode	Instruction	Operation
7F AE	XOR (HL)	A = [A & ~(HL)] | [~A & (HL)]

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 4000	7	n/a	n/a
Rabbit 5000	7	7	5

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•		•	

Description

Performs an exclusive OR operation between A and the data whose address is in HL. The result is stored in A.

The Rabbit 4000/5000 assemblers view “XOR A,(HL)” and “XOR (HL)” as equivalent instructions.

The opcode for this instruction is different than the same instruction in the Rabbit 2000, 3000 and 3000A.

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.

Exclusive OR

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

Opcode	Instruction	Operation
DD AE *d*	XOR (IX+*d*)	A = [A & ~(IX + *d*)] | [~A & (IX + *d*)]
FD AE *d*	XOR (IY+*d*)	A = [A & ~(IY + *d*)] | [~A & (IY + *d*)]

Clocks	8-Bit Access	16-Bit Unaligned	16-Bit Aligned
Rabbit 2000/3000/4000	9	n/a	n/a
Rabbit 5000	10	9	8

Flags				ALTD			IOI/IOE	
S	Z	L/V	C	F	R	SP	S	D
•	•	L	0	•	•		•	

Description

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

- the sum of IX and the 8-bit signed displacement *d*, or
- the sum of IY and *d*

The result is stored in A.

The Rabbit 4000/5000 assemblers view “XOR A,(IX+d)” and “XOR (IX+d)” as equivalent instructions. The same is true for “XOR A,(IY+d)” and “XOR (IY+d).”

Example

If the sum of IX and *d* is 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 (IX+d)
```

would result in the byte in A becoming 1100 0110.

Chapter 4. Quick Reference Guide

This chapter contains an abbreviated description of each Rabbit instruction. The instruction nmemonics are listed alphabetically, each one linking to its full description. For instructions with identical nmemonics, the first entry is the information for the Rabbit 2000 and Rabbit 3000 instruction; the second entry is the Rabbit 4000 instruction.

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 on page 1.
- **`A`**: How the ALTD prefix affects the instruction. For more information, please see Table 3 on page 2.
- **`I`**: How the IOI or IOE prefixes affect the instruction. For more information, please see Table 4 on page 2. 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 2 on page 2.
- **`Operation`**: A symbolic representation of the operation performed.

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Opcode byte 5	Opcode byte 6	AD	IO	S	Z	PV	C	Operation
ADC A,(HL)	8E						fr	s	*	*	V	*	A = A + (HL) + CF
ADC A,(HL)	7F	8E					fr	s	*	*	V	*	A = A + (HL) + CF
ADC A,(IX+d)	DD	8E	----d---				fr	s	*	*	V	*	A = A + (IX+d) + CF
ADC A,(IY+d)	FD	8E	----d---				fr	s	*	*	V	*	A = A + (IY+d) + CF
ADC A,n	CE	----n---					fr		*	*	V	*	A = A + n + CF
ADC A,r	10001-r-						fr		*	*	V	*	A = A + r + CF
ADC A,r	7F	10001-r-					fr		*	*	V	*	A = A + r + CF
ADC HL,ss	ED	01ss1010					fr		*	*	V	*	HL = HL + ss + CF
ADD A,(HL)	86						fr	s	*	*	V	*	A = A + (HL)
ADD A,(HL)	7F	86					fr	s	*	*	V	*	A = A + (HL)
ADD A,(IX+d)	DD	86	----d---				fr	s	*	*	V	*	A = A + (IX+d)
ADD A,(IY+d)	FD	86	----d---				fr	s	*	*	V	*	A = A + (IY+d)
ADD A,n	C6	----n---					fr		*	*	V	*	A = A + n
ADD A,r	10000-r-						fr		*	*	V	*	A = A + r
ADD A,r	7F	10000-r-					fr		*	*	V	*	A = A + r

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Opcode byte 5	Opcode byte 6	AD	IO	S	Z	PV	C	Operation
ADD HL,ss	00ss1001						fr		-	-	-	*	HL = HL + ss
ADD IX,xx	DD	00xx1001							-	-	-	*	IX = IX + xx
ADD IY,yy	FD	00yy1001							-	-	-	*	IY = IY + yy
ADD SP,d	27	----d---							-	-	-	*	SP = SP + d
ADD HL,JK	65						fr		-	-	-	*	HL = HL + JK
ADD JKHL,BCDE	ED	C6					fr		-	-	-	*	JKHL = JKHL + BCDE
ALTD	76								-	-	-	-	alternate register destination for next instruction
AND (HL)	A6						fr	s	*	*	P	0	A = A & (HL)
AND (HL)	7F	A6					fr	s	*	*	P	0	A = A & (HL)
AND (IX+d)	DD	A6	----d---				fr	s	*	*	P	0	A = A & (IX+d)
AND (IY+d)	FD	A6	----d---				fr	s	*	*	P	0	A = A & (IY+d)
AND HL,DE	DC						fr		*	*	P	0	HL = HL & DE
AND IX,DE	DD	DC							*	*	P	0	IX = IX & DE
AND IY,DE	FD	DC							*	*	P	0	IY = IY & DE
AND JKHL,BCDE	ED	E6					fr		*	*	P	0	JKHL = JKHL & BCDE
AND n	E6	----n---					fr		*	*	P	0	A = A & n
AND r	10100-r-						fr		*	*	P	0	A = A & r
AND r	7F	10100-r-					fr		*	*	P	0	A = A & r
BIT b,(HL)	CB	01-b-110					fr	s	-	*	-	-	(HL) & bit
BIT b,(IX+d)	DD	CB	----d---	01-b-110				s	-	*	-	-	(IX+d) & bit
BIT b,(IY+d)	FD	CB	----d---	01-b-110				s	-	*	-	-	(IY+d) & bit
BIT b,r	CB	01-b--r-					fr		-	*	-	-	r & bit
BOOL HL	CC						fr		*	*	0	0	if (HL != 0) HL = 1
BOOL IX	DD	CC							*	*	0	0	if (IX != 0) IX = 1
BOOL IY	FD	CC							*	*	0	0	if (IY != 0) IY = 1
CALL mn	CD	----n---	----m---						-	-	-	-	(SP-1) = PCH; (SP-2) = PCL; PC = mn; SP = SP-2
CALL (HL)	ED	EA							-	-	-	-	(SP-1) = PCH; (SP-2) = PCL; PC = HL; SP = SP-2
CALL (IX)	DD	EA							-	-	-	-	(SP-1) = PCH; (SP-2) = PCL; PC = IX; SP = SP-2
CALL (IY)	FD	EA							-	-	-	-	(SP-1) = PCH; (SP-2) = PCL; PC = IY; SP = SP-2
CBM n	ED	00	----n---					d	-	-	-	-	tmp = [(HL) & ~n] | [a & n]; (HL) = tmp; (DE) = tmp (only (DE) affected by IOI or IOE)
CCF	3F						f		-	-	-	*	CF = ~CF
CLR HL	BF						r		-	-	-	-	HL = 0
CONVC pp	ED	00pp1110							-	-	-	-	convert pp to physical code address

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Opcode byte 5	Opcode byte 6	AD	IO	S	Z	PV	C	Operation
CONVD pp	ED	00pp1111							-	-	-	-	convert pp to physical data address
COPY	ED	80							-	-	*	-	(PY) = (PX); BC = BC-1; PY = PY+1; PX = PX+1; repeat while {BC != 0}
COPYR	ED	88							-	-	*	-	(PY) = (PX); BC = BC-1; PY = PY-1; PX = PX-1; repeat while {BC != 0}
CP (HL)	BE						f	s	*	*	V	*	A - (HL)
CP (HL)	7F	BE					f	s	*	*	V	*	A - (HL)
CP HL,d	48	----d---					f		*	*	V	*	HL - d (d sign-extened to 16 bits)
CP HL,DE	ED	48					f		*	*	V	*	HL - DE
CP (IX+d)	DD	BE	----d---				f	s	*	*	V	*	A - (IX+d)
CP (IY+d)	FD	BE	----d---				f	s	*	*	V	*	A - (IY+d)
CP JKHL,BCDE	ED	58					f		*	*	V	*	JKHL - BCDE
CP n	FE	----n---					f		*	*	V	*	A - n
CP r	10111-r-						f		*	*	V	*	A - r
CP r	7F	10111-r-					f		*	*	V	*	A - r
CPL	2F						r		-	-	-	-	A = ~A
DEC (HL)	35						f	b	*	*	V	*	(HL) = (HL) - 1
DEC (IX+d)	DD	35	----d---				f	b	*	*	V	*	(IX+d) = (IX+d) -1
DEC (IY+d)	FD	35	----d---				f	b	*	*	V	*	(IY+d) = (IY+d) -1
DEC IX	DD	2B							-	-	-	-	IX = IX - 1
DEC IY	FD	2B							-	-	-	-	IY = IY - 1
DEC r	00-r-101						fr		*	*	V	*	r = r - 1
DEC ss	00ss1011						r		-	-	-	-	ss = ss - 1
DJNZ label	10	--e-					r		-	-	-	-	B = B-1; if {B != 0} PC = PC + j
DWJNZ label	ED	10	-e-				r		-	-	-	-	BC = BC-1; if {BC != 0} PC = PC + e
EX AF,AF’	08								-	-	-	-	AF <-> AF’
EX BC,HL	B3						s		-	-	-	-	if (!ALTD) then BC <-> HL else BC <-> HL’
EX BC’,HL	ED	74					s		-	-	-	-	if (!ALTD) then BC’ <-> HL else BC’ <-> HL’
EX DE,HL	EB						s		-	-	-	-	if (!ALTD) then DE <-> HL else DE <-> HL’
EX DE’,HL	E3						s		-	-	-	-	if (!ALTD) then DE’ <-> HL else DE’ <-> HL’
EX JK,HL	B9								-	-	-	-	if (!ALTD) then JK <-> HL else JK <-> HL’
EX JK’,HL	ED	7C					s		-	-	-	-	if (!ALTD) then JK’ <-> HL else JK’ <-> HL’
EX JKHL,BCDE	B4								-	-	-	-	JKHL <-> BCDE
EX (SP),HL	ED	54					r		-	-	-	-	H <-> (SP+1); L <-> (SP)
EX (SP),IX	DD	E3							-	-	-	-	IXH <-> (SP+1); IXL <-> (SP)

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Opcode byte 5	Opcode byte 6	AD	IO	S	Z	PV	C	Operation
EX (SP),IY	FD	E3							-	-	-	-	IYH <-> (SP+1); IYL <-> (SP)
EXP	ED	D9							-	-	-	-	PW <-> PW'; PX <-> PX'; PY <-> PY'; PZ <-> PZ'
EXX	D9								-	-	-	-	BC <-> BC'; DE <-> DE'; HL <-> HL'
FLAG cc,HL	ED	110cc100							-	-	-	-	if (cc) then HL = 1 else HL = 0
FSYSCALL	ED	55							-	-	-	-	(SP - 1) = PChigh; (SP - 2) = PClow; (SP - 3) = SU; SP = SP - 3; PC = {IIR,011000000}; SU = {SU[5:0],00}
IBOX A	ED	12					r		-	-	-	-	A = inverse sbox(A)
IDET	5B								-	-	-	-	Performs 'LD E,E'' But if (EDMR && SU[0]) then the System Violation interrupt flag is set
INC (HL)	34						f	b	*	*	V	*	(HL) = (HL) + 1
INC (IX+d)	DD	34	----d---				f	b	*	*	V	*	(IX+d) = (IX+d) + 1
INC (IY+d)	FD	34	----d---				f	b	*	*	V	*	(IY+d) = (IY+d) + 1
INC IX	DD	23							-	-	-	-	IX = IX + 1
INC IY	FD	23							-	-	-	-	IY = IY + 1
INC r	00-r-100						fr		*	*	V	*	r = r + 1
INC ss	00ss0011						r		-	-	-	-	ss = ss + 1
IOE	DB								-	-	-	-	I/O external prefix
IOI	D3								-	-	-	-	I/O internal prefix
IPSET 0	ED	46							-	-	-	-	IP = {IP[5:0], 00}
IPSET 1	ED	56							-	-	-	-	IP = {IP[5:0], 01}
IPSET 2	ED	4E							-	-	-	-	IP = {IP[5:0], 10}
IPSET 3	ED	5E							-	-	-	-	IP = {IP[5:0], 11}
IPRES	ED	5D							-	-	-	-	IP = {IP[1:0], IP[7:2]}
JP cx,mn	101cx010	----n---	----m---						-	-	-	-	if {cx} PC = mn
JP (HL)	E9								-	-	-	-	PC = HL
JP (IX)	DD	E9							-	-	-	-	PC = IX
JP (IY)	FD	E9							-	-	-	-	PC = IY
JP f,mn	11-f-010	----n---	----m---						-	-	-	-	if {f} PC = mn
JP mn	C3	----n---	----m---						-	-	-	-	PC = mn
JR cc,label	001cc000	--e-							-	-	-	-	if {cc} PC = PC + e
JR cx,label	101cx000	-e-							-	-	-	-	if {cx} PC = PC + e
JR label	18	--e-							-	-	-	-	PC = PC + e
JRE cc,label	ED	110cc011	-(ee)low-	-(ee)high-					-	-	-	-	if {cc} PC = PC + ee

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Opcode byte 5	Opcode byte 6	AD	IO	S	Z	PV	C	Operation
JRE cx,label	ED	101cx011	-(ee)low-	-(ee)high-					-	-	-	-	if {cx} PC = PC + ee
JRE label	98	-(ee)low-	-(ee)high-						-	-	-	-	PC = PC + ee
LCALL xpc,mn	CF	----n---	----m---	--xpc---					-	-	-	-	(SP-1) = XPCl; (SP-2) = PCH; (SP-3) = PCL; XPCl = xpc; XPCh = 0; PC = mn; SP = SP-3
LD A,EIR	ED	57					r		*	*	-	-	A = EIR
LD A,IIR	ED	5F					r		*	*	-	-	A = IIR
LD A,HTR	ED	50					r		-	-	-	-	A = HTR
LD A,r	01111rna						r		-	-	-	-	A = r (only for r != A)
LD A,(BC)	0A						r	s	-	-	-	-	A = (BC)
LD A,(DE)	1A						r	s	-	-	-	-	A = (DE)
LD A,(IX+A)	DD	06						s	-	-	-	-	A = (IX+A)
LD A,(IY+A)	FD	06						s	-	-	-	-	A = (IY+A)
LD A,(ps+d)	10ps1101	----d---					r		-	-	-	-	A = (ps+d)
LD A,(ps+HL)	10ps1011						r		-	-	-	-	A = (ps+HL)
LD A,(mn)	3A	----n---	----m---				r	s	-	-	-	-	A = (mn)
LD A,XPC	ED	77					r		-	-	-	-	A = XPCl
LD BC,HL	91						r		-	-	-	-	BC = HL
LD BCDE,(HL)	DD	1A					r		-	-	-	-	E = (HL); D = (HL+1); C = (HL+2); B = (HL+3)
LD BCDE,(IX+d)	DD	CE	----d---				r		-	-	-	-	E = (IX+d); D = (IX+d+1); C = (IX+d+2); B = (IX+d+3)
LD BCDE,(IY+d)	DD	DE	----d---				r		-	-	-	-	E = (IY+d); D = (IY+d+1); C = (IY+d+2); B = (IY+d+3)
LD BCDE,(mn)	93	----n---	----m---				r		-	-	-	-	E = (mn); D = (mn+1); C = (mn+2); B = (mn+3)
LD BCDE,(ps+d)	DD	00ps1110	----d---				r		-	-	-	-	E = (ps+d); D = (ps+d+1); C = (ps+d+2); B = (ps+d+3)
LD BCDE,(ps+HL)	DD	00ps1100					r		-	-	-	-	E = (ps+HL); D = (ps+HL+1); C = (ps+HL+2); B = (ps+HL+3)
LD BCDE,(SP+HL)	DD	FE					r		-	-	-	-	E = (SP+HL); D = (SP+HL+1); C = (SP+HL+2); B = (SP+HL+3)
LD BCDE,(SP+n)	DD	EE	----n---				r		-	-	-	-	E = (SP+n); D = (SP+n+1); C = (SP+n+2); B = (SP+n+3)
LD BCDE,n	A3	----n---					r		-	-	-	-	E = n; D = 0; C = 0; B = 0
LD BCDE,ps	DD	11ps1101					r		-	-	-	-	E = ps0; D = ps1; C = ps2; B = 00/FF
LD DE,HL	B1						r		-	-	-	-	DE = HL
LD HL,(ps+BC)	ED	00ps0110					r		-	-	-	-	L = (ps+BC); H = (ps+BC+1)
LD HL,(ps+d)	10ps0101	----d---					r		-	-	-	-	L = (ps+d); H = (ps+d+1)
LD HL,(SP+HL)	ED	FE					r		-	-	-	-	L = (SP+HL); H = (SP+HL+1)
LD HL,BC	81						r		-	-	-	-	HL = BC

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Opcode byte 5	Opcode byte 6	AD	IO	S	Z	PV	C	Operation
LD HL,DE	A1						r		-	-	-	-	HL = DE
LD HL,LXPC	9F						r		-	-	-	-	HL = LXPC
LD HTR,A	ED	40							-	-	-	-	HTR = A
LD JK,(mn)	99	----n---	----m---				r	s	-	-	-	-	K = (mn); J = (mn+1)
LD JK,mn	A9	----n---	----m---				r		-	-	-	-	K = n; J = m
LD JKHL,(HL)	FD	1A					r		-	-	-	-	L = (HL); H = (HL+1); K = (HL+2); J = (HL+3)
LD JKHL,(IX+d)	FD	CE	----d---				r		-	-	-	-	L = (IX+d); H = (IX+d+1); K = (IX+d+2); J = (IX+d+3)
LD JKHL,(IY+d)	FD	DE	----d---				r		-	-	-	-	L = (IY+d); H = (IY+d+1); K = (IY+d+2); J = (IY+d+3)
LD JKHL,(mn)	94	----n---	----m---				r		-	-	-	-	L = (mn); H = (mn+1); K = (mn+2); J = (mn+3)
LD JKHL,(ps+d)	FD	00ps1110	----d---				r		-	-	-	-	L = (ps+d); H = (ps+d+1); K = (ps+d+2); J = (ps+d+3)
LD JKHL,(ps+HL)	FD	00ps1100					r		-	-	-	-	L = (ps+HL); H = (ps+HL+1); K = (ps+HL+2); J = (ps+HL+3)
LD JKHL,(SP+HL)	FD	FE					r		-	-	-	-	L = (SP+HL); H = (SP+HL+1); K = (SP+HL+2); J = (SP+HL+3)
LD JKHL,(SP+n)	FD	EE	----n---				r		-	-	-	-	L = (SP+n); H = (SP+n+1); K = (SP+n+2); J = (SP+n+3)
LD JKHL,d	A4	----d---					r		-	-	-	-	L = d; H = 0; K = 0; J = 0
LD JKHL,ps	FD	11ps1101					r		-	-	-	-	L = ps0; H = ps1; K = ps2; J = 00/FF
LD (BC),A	02							d	-	-	-	-	(BC) = A
LD (DE),A	12							d	-	-	-	-	(DE) = A
LD (HL),n	36	----n---						d	-	-	-	-	(HL) = n
LD (HL),r	01110-r-							d	-	-	-	-	(HL) = r
LD (HL+d),HL	DD	F4	----d---					d	-	-	-	-	(HL+d) = L; (HL+d+1) = H
LD (IX+d),HL	F4	----d---						d	-	-	-	-	(IX+d) = L; (IX+d+1) = H
LD (IX+d),n	DD	36	----d---	----n---				d	-	-	-	-	(IX+d) = n
LD (IX+d),r	DD	01110-r-	----d---					d	-	-	-	-	(IX+d) = r
LD (IY+d),HL	FD	F4	----d---					d	-	-	-	-	(IY+d) = L; (IY+d+1) = H
LD (IY+d),n	FD	36	----d---	----n---				d	-	-	-	-	(IY+d) = n
LD (IY+d),r	FD	01110-r-	----d---					d	-	-	-	-	(Iy+d) = r
LD (mn),A	32	----n---	----m---					d	-	-	-	-	(mn) = A
LD (mn),HL	22	----n---	----m---					d	-	-	-	-	(mn) = L; (mn+1) = H
LD (mn),IX	DD	22	----n---	----m---				d	-	-	-	-	(mn) = IXL; (mn+1) = IXH
LD (mn),IY	FD	22	----n---	----m---				d	-	-	-	-	(mn) = IYL; (mn+1) = IYH
LD (mn),ss	ED	01ss0011	----n---	----m---				d	-	-	-	-	(mn) = ssl; (mn+1) = ssh
LD (HL),BCDE	DD	1B							-	-	-	-	(HL) = E; (HL+1) = D; (HL+2) = C; (HL+3) = B
LD (HL),JKHL	FD	1B							-	-	-	-	(HL) = L; (HL+1) = H; (HL+2) = K; (HL+3) = J

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Opcode byte 5	Opcode byte 6	AD	IO	S	Z	PV	C	Operation
LD (IX+d),BCDE	DD	CF	----d---						-	-	-	-	(IX+d) = E; (IX+d+1) = D; (IX+d+2) = C; (IX+d+3) = B
LD (IX+d),JKHL	FD	CF	----d---						-	-	-	-	(IX+d) = L; (IX+d+1) = H; (IX+d+2) = K; (IX+d+3) = J
LD (IY+d),BCDE	DD	DF	----d---						-	-	-	-	(IY+d) = E; (IY+d+1) = D; (IY+d+2) = C; (IY+d+3) = B
LD (IY+d),JKHL	FD	DF	----d---						-	-	-	-	(IY+d) = L; (IY+d+1) = H; (IY+d+2) = K; (IY+d+3) = J
LD (mn),BCDE	83	----n---	----m---						-	-	-	-	(mn) = E; (mn+1) = D; (mn+2) = C; (mn+3) = B
LD (mn),JK	89	----n---	----m---					d	-	-	-	-	(mn) = K; (mn+1) = J
LD (mn),JKHL	84	----n---	----m---						-	-	-	-	(mn) = L; (mn+1) = H; (mn+2) = K; (mn+3) = J
LD (pd+BC),HL	ED	00pd0111							-	-	-	-	(pd+BC) = L; (pd+BC+1) = H
LD (pd+d),A	10pd1110	----d---							-	-	-	-	(pp+d) = A
LD (pd+d),BCDE	DD	00pd1111	----d---						-	-	-	-	(pd+d) = E; (pd+d+1) = D; (pd+d+2) = C; (pd+d+3) = B
LD (pd+d),HL	10pd0110	----d---							-	-	-	-	(pd+d) = L; (pd+d+1) = H
LD (pd+d),JKHL	FD	00pd1111	----d---						-	-	-	-	(pd+d) = L; (pd+d+1) = H; (pd+d+2) = K; (pd+d+3) = J
LD (pd+d),ps	6D	pspd1001	----d---						-	-	-	-	(pd+d) = ps0; (pd+d+1) = ps1; (pd+d+2) = ps2; (pd+d+3) = ps3
LD (pd+d),rr	6D	rrpd0001	----d---						-	-	-	-	(pd+d) = rrl; (pd+d+1) = rrh
LD (pd+HL),A	10pd1100								-	-	-	-	(pd+HL) = A
LD (pd+HL),BCDE	DD	00pd1101							-	-	-	-	(pd+HL) = E; (pd+HL+1) = D; (pd+HL+2) = C; (pd+HL+3) = B
LD (pd+HL),JKHL	FD	00pd1101							-	-	-	-	(pd+HL) = L; (pd+HL+1) = H; (pd+HL+2) = K; (pd+HL+3) = J
LD (pd+HL),ps	6D	pspd1011							-	-	-	-	(pd+HL) = ps0; (pd+HL+1) = ps1; (pd+HL+2) = ps2; (pd+HL+3) = ps3
LD (pd+HL),rr	6D	rrpd0011							-	-	-	-	(pd+HL) = rrl; (pd+HL+1) = rrh
LD (SP+n),HL	D4	----n---							-	-	-	-	(SP+n) = L; (SP+n+1) = H
LD (SP+n),IX	DD	D4	----n---						-	-	-	-	(SP+n) = IXL; (SP+n+1) = IXH
LD (SP+n),IY	FD	D4	----n---						-	-	-	-	(SP+n) = IYL; (SP+n+1) = IYH
LD dd,(mn)	ED	01dd1011	----n---	----m---			r	s	-	-	-	-	ddl = (mn); ddh = (mn+1)
LD dd',BC	ED	01dd1001							-	-	-	-	dd' = BC (dd': 00-BC', 01-DE', 10-HL')
LD dd',DE	ED	01dd0001							-	-	-	-	dd' = DE (dd': 00-BC', 01-DE', 10-HL')
LD dd,mn	00dd0001	----n---	----m---				r		-	-	-	-	dd = mn
LD HL,(mn)	2A	----n---	----m---				r	s	-	-	-	-	L = (mn); H = (mn+1)
LD HL,(HL+d)	DD	E4	----d---				r	s	-	-	-	-	L = (HL+d); H = (HL+d+1)
LD HL,(IX+d)	E4	----d---					r	s	-	-	-	-	L = (IX+d); H = (IX+d+1)
LD HL,(IY+d)	FD	E4	----d---				r	s	-	-	-	-	L = (IY+d); H = (IY+d+1)
LD HL,(SP+n)	C4	----n---					r		-	-	-	-	L = (SP+n); H = (SP+n+1)
LD HL,IX	DD	7C					r		-	-	-	-	HL = IX
LD HL,IY	FD	7C					r		-	-	-	-	HL = IY

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Opcode byte 5	Opcode byte 6	AD	IO	S	Z	PV	C	Operation
LD EIR,A	ED	47							-	-	-	-	I = A
LD IX,(mn)	DD	2A	----n---	----m---				s	-	-	-	-	IXL = (mn); IXH = (mn+1)
LD IX,(SP+n)	DD	C4	----n---						-	-	-	-	IXL = (SP+n); IXH = (SP+n+1)
LD IX,HL	DD	7D							-	-	-	-	IX = HL
LD IX,mn	DD	21	----n---	----m---					-	-	-	-	IX = mn
LD IY,(mn)	FD	2A	----n---	----m---				s	-	-	-	-	IYL = (mn); IYH = (mn+1)
LD IY,(SP+n)	FD	C4	----n---						-	-	-	-	IYL = (SP+n); IYH = (SP+n+1)
LD IY,HL	FD	7D							-	-	-	-	IY = HL
LD IY,mn	FD	21	----n---	----m---					-	-	-	-	IY = mn
LD pd,(ps+d)	6D	pdps1000	----d---						-	-	-	-	pd0 = (ps+d); pd1 = (ps+d+1); pd2 = (ps+d+2); pd3 = (ps+d+3)
LD pd,(ps+HL)	6D	pdps1010							-	-	-	-	pd0 = (ps+HL); pd1 = (ps+HL+1); pd2 = (ps+HL+2); pd3 = (ps+HL+3)
LD pd,(SP+n)	ED	00pd0100	----n---						-	-	-	-	pd0 = (SP+n); pd1 = (SP+n+1); pd2 = (SP+n+2); pd3 = (SP+n+3)
LD pd,(HTR+HL)	ED	00pd0001							-	-	-	-	pd0 = (HTR+HL); pd1 = (HTR+HL+1); pd2 = (HTR+HL+2); pd3 = (HTR+HL+3)
LD pd,BCDE	DD	10pd1101							-	-	-	-	pd0 = E; pd1 = D; pd2 = C
LD pd,JKHL	FD	10pd1101							-	-	-	-	pd0 - L; pd1 = H; pd2 = K
LD pd,klmn	ED	00pd1100	----n---	----m---	----l---	----k---			-	-	-	-	pd0 = n; pd1 = m; pd2 = l; pd3 = k
LD pd,ps+d	6D	pdps1100	----d---						-	-	-	-	pd = ps + d
LD pd,ps+HL	6D	pdps1110							-	-	-	-	pd = ps + HL
LD rr,(ps+d)	6D	rrps0000	----d---				r		-	-	-	-	rrl = (ps+d); rrh = (ps+d+1)
LD rr,(ps+HL)	6D	rrps0010					r		-	-	-	-	rrl = (ps+HL); rrh = (ps+HL+1)
LD r,(HL)	01-r-110						r	s	-	-	-	-	r = (HL)
LD r,(IX+d)	DD	01-r-110	----d---				r	s	-	-	-	-	r = (IX+d)
LD r,(IY+d)	FD	01-r-110	----d---				r	s	-	-	-	-	r = (IY+d)
LD IIR,A	ED	4F							-	-	-	-	R = A
LD r,n	00-r-110	----n---					r		-	-	-	-	r = n
LD r,g	01-r--r’						r		-	-	-	-	r = g
LD r,g	7F	01-r--r’					r		-	-	-	-	r = g
LD SP,HL	F9								-	-	-	-	SP = HL
LD SP,IX	DD	F9							-	-	-	-	SP = IX
LD SP,IY	FD	F9							-	-	-	-	SP = IY
LD (SP+HL),BCDE	DD	FF							-	-	-	-	(SP+HL) = E; (SP+HL+1) = D; (SP+HL+2) = C; (SP+HL+3) = B
LD (SP+HL),JKHL	FD	FF							-	-	-	-	(SP+HL) = L; (SP+HL+1) = H; (SP+HL+2) = K; (SP+HL+3) = J
LD (SP+n),BCDE	DD	EF	----n---						-	-	-	-	(SP+n) = E; (SP+n+1) = D; (SP+n+2) = C; (SP+n+3) = B

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Opcode byte 5	Opcode byte 6	AD	IO	S	Z	PV	C	Operation
LD (SP+n),JKHL	FD	EF	----n---						-	-	-	-	(SP+n) = L; (SP+n+1) = H; (SP+n+2) = K; (SP+n+3) = J
LD (SP+n),ps	ED	00ps0101	----n---						-	-	-	-	(SP+n) = ps0; (SP+n+1) = ps1; (SP+n+2) = ps2; (SP+n+3) = ps3
LD XPC,A	ED	67							-	-	-	-	XPCl = A; XPCh = 0
LD LXPC,HL	97								-	-	-	-	XPCl = L; XPCh = H
LDD	ED	A8						d	-	-	*	-	(DE) = (HL); BC = BC-1; DE = DE-1; HL = HL-1
LDDR	ED	B8						d	-	-	*	-	(DE) = (HL); BC = BC-1; DE = DE-1; HL = HL-1; repeat while {BC != 0}
LDDSR	ED	98						d	-	-	*	-	(DE) = (HL); BC = BC-1; HL = HL-1; repeat while {BC != 0}
LDF (lmn),A	8A	----n---	----m---	----l---					-	-	-	-	(lmn) = A
LDF (lmn),BCDE	DD	0B	----n---	----m---	----l---				-	-	-	-	(lmn) = E; (lmn+1) = D; (lmn+2) = C; (lmn+3) = B
LDF (lmn),HL	82	----n---	----m---	----l---					-	-	-	-	(lmn) = L; (lmn+1) = H
LDF (lmn),JKHL	FD	0B	----n---	----m---	----l---				-	-	-	-	(lmn) = L; (lmn+1) = H; (lmn+2) = K; (lmn+3) = J
LDF (lmn),ps	ED	00ps1001	----n---	----m---	----l---				-	-	-	-	(lmn) = ps0; (lmn+1) = ps1; (lmn+2) = ps2; (lmn+3) = ps3
LDF (lmn),rr	ED	00rr1011	----n---	----m---	----l---				-	-	-	-	(lmn) = rrl; (lmn+1) = rrh
LDF A,(lmn)	9A	----n---	----m---	----l---			r		-	-	-	-	A = (lmn)
LDF BCDE,(lmn)	DD	0A	----n---	----m---	----l---				-	-	-	-	E = (lmn); D = (lmn+1); C = (lmn+2); B = (lmn+3)
LDF HL,(lmn)	92	----n---	----m---	----l---			r		-	-	-	-	L = (lmn); H = (lmn+1)
LDF JKHL,(lmn)	FD	0A	----n---	----m---	----l---		r		-	-	-	-	L = (lmn); H = (lmn+1); K = (lmn+2); J = (lmn+3)
LDF pd,(lmn)	ED	00pd1000	----n---	----m---	----l---		r		-	-	-	-	pd0 = (lmn); pd1 = (lmn+1); pd2 = (lmn+2); pd3 = (lmn+3)
LDF rr,(lmn)	ED	00rr1010	----n---	----m---	----l---		r		-	-	-	-	rrl = (lmn); rrh = (lmn+1)
LDI	ED	A0						d	-	-	*	-	(DE) = (HL); BC = BC-1; DE = DE+1; HL = HL+1
LDIR	ED	B0						d	-	-	*	-	(DE) = (HL); BC = BC-1; DE = DE+1; HL = HL+1; repeat while {BC != 0}
LDISR	ED	90						d	-	-	*	-	(DE) = (HL); BC = BC-1; HL = HL+1; repeat while {BC != 0}
LDL pd,DE	DD	10pd1111					r		-	-	-	-	pd = {FFFF,DE}
LDL pd,HL	FD	10pd1111					r		-	-	-	-	pd = {FFFF,HL}
LDL pd,IX	DD	10pd1100					r		-	-	-	-	pd = {FFFF,IX}
LDL pd,IY	FD	10pd1100					r		-	-	-	-	pd = {FFFF,IY}
LDL pd,mn	ED	00pd1101	n	m			r		-	-	-	-	pd = {FFFF,mn}
LDL pd,(SP+n)	ED	00pd0011	n				r		-	-	-	-	pd0 = (SP+n); pd1= (SP+d+1); pd2=FF; pd3=FF
LDP (HL),HL	ED	64							-	-	-	-	(HL) = L; (HL+1) = H. (Addr[19:16] = A[3:0])
LDP (IX),HL	DD	64							-	-	-	-	(IX) = L; (IX+1) = H. (Addr[19:16] = A[3:0])
LDP (IY),HL	FD	64							-	-	-	-	(IY) = L; (IY+1) = H. (Addr[19:16] = A[3:0])
LDP (mn),HL	ED	65	----n---	----m---					-	-	-	-	(mn) = L; (mn+1) = H. (Addr[19:16] = A[3:0])

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Opcode byte 5	Opcode byte 6	AD	IO	S	Z	PV	C	Operation
LDP (mn),IX	DD	65	----n---	----m---					-	-	-	-	(mn) = IXL; (mn+1) = IXH. (Addr[19:16] = A[3:0])
LDP (mn),IY	FD	65	----n---	----m---					-	-	-	-	(mn) = IYL; (mn+1) = IYH. (Addr[19:16] = A[3:0])
LDP HL,(HL)	ED	6C							-	-	-	-	L = (HL); H = (HL+1). (Addr[19:16] = A[3:0])
LDP HL,(IX)	DD	6C							-	-	-	-	L = (IX); H = (IX+1). (Addr[19:16] = A[3:0])
LDP HL,(IY)	FD	6C							-	-	-	-	L = (IY); H = (IY+1). (Addr[19:16] = A[3:0])
LDP HL,(mn)	ED	6D	----n---	----m---					-	-	-	-	L = (mn); H = (mn+1). (Addr[19:16] = A[3:0])
LDP IX,(mn)	DD	6D	----n---	----m---					-	-	-	-	IXL = (mn); IXH = (mn+1). (Addr[19:16] = A[3:0])
LDP IY,(mn)	FD	6D	----n---	----m---					-	-	-	-	IYL = (mn); IYH = (mn+1). (Addr[19:16] = A[3:0])
LJP xpc,mn	C7	----n---	----m---	--xpc--					-	-	-	-	XPCl = xpc; XPCh = 0; PC = mn
LLCALL lxpc,mn	8F	----n---	----m---	---xpl---	---xph---				-	-	-	-	(SP-1) = XPCh; (SP-2) = XPCl; (SP-3) = PCH; (SP-4) = PCL; XPCl = xpl; XPCh = xph; PC = mn; SP = SP-4
LLJP cc,lxpc,mn	ED	110cc010	----n---	----m---	---xpl--	---xph--			-	-	-	-	if {cc} XPCl = xpl; XPCh = xph; PC = mn
LLJP cx,lxpc,mn	ED	101cx010	----n---	----m---	---xpl--	---xph--			-	-	-	-	if {cx} XPCl = xpl; XPCh = xph; PC = mn
LLJP lxpc,mn	87	----n---	----m---	---xpl---	---xph---				-	-	-	-	XPCl = xpl; XPCh = xph; PC = mn
LLRET	ED	8B							-	-	-	-	PCL = (SP); PCH = (SP+1); XPCl = (SP+2); XPCh = (SP+4); SP = SP+4
LRET	ED	45							-	-	-	-	PCL = (SP); PCH = (SP+1); XPCl = (SP+2); XPCh = 0; SP = SP+3
LSDDR	ED	D8						s	-	-	*	-	(DE) = (HL); BC = BC-1; DE = DE-1; repeat while {BC != 0}
LSDR	ED	F8						s	-	-	*	-	(DE) = (HL); BC = BC-1; DE = DE-1; HL = HL-1; repeat while {BC != 0}
LSIDR	ED	D0						s	-	-	*	-	(DE) = (HL); BC = BC-1; DE = DE+1; repeat while {BC != 0}
LSIR	ED	F0						s	-	-	*	-	(DE) = (HL); BC = BC-1; DE = DE+1; HL = HL+1; repeat while {BC != 0}
MUL	F7								-	-	-	-	HL:BC = BC * DE
MULU	A7								-	-	-	-	HL:BC = BC * DE (unsigned)
NEG	ED	44					fr		*	*	V	*	A = 0 - A
NEG BCDE	DD	4D							*	*	V	*	BCDE = 0 - BCDE
NEG HL	4D						fr		*	*	V	*	HL = 0 - HL
NEG JKHL	FD	4D							*	*	V	*	JKHL = 0 - JKHL
NOP	00								-	-	-	-	No operation
OR (HL)	B6						fr	s	*	*	P	0	A = A | (HL)
OR (HL)	7F	B6					fr	s	*	*	P	0	A = A | (HL)
OR (IX+d)	DD	B6	----d---				fr	s	*	*	P	0	A = A | (IX+d)
OR (IY+d)	FD	B6	----d---				fr	s	*	*	P	0	A = A | (IY+d)
OR HL,DE	EC						fr		*	*	P	0	HL = HL | DE
OR IX,DE	DD	EC							*	*	P	0	IX = IX | DE

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Opcode byte 5	Opcode byte 6	AD	IO	S	Z	PV	C	Operation
OR IY,DE	FD	EC							*	*	P	0	IY = IY | DE
OR JKHL,BCDE	ED	F6					fr		*	*	P	0	JKHL = JKHL | BCDE
OR n	F6	----n---					fr		*	*	P	0	A = A | n
OR r	10110-r-						fr		*	*	P	0	A = A | r
OR A	B7						fr		*	*	P	0	A = A | A
OR r	7F	10110-r-					fr		*	*	P	0	A = A | r
POP IP	ED	7E							-	-	-	-	IP = (SP); SP = SP+1
POP IX	DD	E1							-	-	-	-	IXL = (SP); IXH = (SP+1); SP = SP+2
POP IY	FD	E1							-	-	-	-	IYL = (SP); IYH = (SP+1); SP = SP+2
POP BCDE	DD	F1					r		-	-	-	-	E = (SP); D = (SP+1); C = (SP+2); B = (SP+3); SP = SP+4
POP JKHL	FD	F1					r		-	-	-	-	L = (SP); H = (SP+1); K = (SP+2); J = (SP+3); SP = SP+4
POP SU	ED	6E							-	-	-	-	SU = (SP); SP = SP+1
POP zz	11zz0001						r		-	-	-	-	zzl = (SP); zzh = (SP+1); SP = SP+2
POP pd	ED	11pd0001					r		-	-	-	-	pd0 = (SP); pd1 = (SP+1); pd2 = (SP+2); pd3 = (SP+3); SP = SP+4
PUSH IP	ED	76							-	-	-	-	(SP-1) = IP; SP = SP-1
PUSH IX	DD	E5							-	-	-	-	(SP-1) = IXH; (SP-2) = IXL; SP = SP-2
PUSH IY	FD	E5							-	-	-	-	(SP-1) = IYH; (SP-2) = IYL; SP = SP-2
PUSH BCDE	DD	F5							-	-	-	-	(SP-1) = B; (SP-2) = C; (SP-3) = D; (SP-4) = E; SP = SP-4
PUSH JKHL	FD	F5							-	-	-	-	(SP-1) = J; (SP-2) = K; (SP-3) = H; (SP-4) = L; SP = SP-4
PUSH mn	ED	A5	----n---	----m---					-	-	-	-	(SP-1) = m; (SP-2) = n; SP = SP - 2
PUSH ps	ED	11ps0101							-	-	-	-	(SP-1) = ps3; (SP-2) = ps2; (SP-3) = ps1; (SP-4) = ps0; SP = SP-4
PUSH SU	ED	66							-	-	-	-	(SP-1) = SU; SP = SP-1
PUSH zz	11zz0101								-	-	-	-	(SP-1) = zzh; (SP-2) = zzl; SP = SP-2
RDMODE	ED	7F							-	-	-	*	CF = SU[0]
RES b,(HL)	CB	10-b-110						d	-	-	-	-	(HL) = (HL) & ~bit
RES b,(IX+d)	DD	CB	----d---	10-b-110				d	-	-	-	-	(IX+d) = (IX+d) & ~bit
RES b,(IY+d)	FD	CB	----d---	10-b-110				d	-	-	-	-	(IY+d) = (IY+d) & ~bit
RES b,r	CB	10-b--r-					r		-	-	-	-	r = r & ~bit
RET	C9								-	-	-	-	PCL = (SP); PCH = (SP+1); SP = SP+2
RET f	11-f-000								-	-	-	-	if {f} PCL = (SP); PCH = (SP+1); SP = SP+2
RETI	ED	4D							-	-	-	-	IP = (SP); PCL = (SP+1); PCH = (SP+2); SP = SP+3
RL (HL)	CB	16					f	b	*	*	P	*	{CF,(HL)} = {(HL),CF}
RL (IX+d)	DD	CB	----d---	16			f	b	*	*	P	*	{CF,(IX+d)} = {(IX+d),CF}

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Opcode byte 5	Opcode byte 6	AD	IO	S	Z	PV	C	Operation
RL (IY+d)	FD	CB	----d---	16			f	b	*	*	P	*	{CF,(IY+d)} = {(IY+d),CF}
RL DE	F3						fr		*	*	P	*	{CF,DE} = {DE,CF}
RL r	CB	00010-r-					fr		*	*	P	*	{CF,r} = {r,CF}
RL bb,BCDE	DD	011010bb					fr		*	*	P	*	{CF,BCDE} = {BCDE,CF}; bb=bb-1; repeat while bb!=0
RL bb,JKHL	FD	011010bb					fr		*	*	P	*	{CF,JKHL} = {JKHL,CF}; bb=bb-1; repeat while bb!=0
RL BC	62						fr		*	*	P	*	{CF,BC} = {BC,CF}
RL HL	42						fr		*	*	P	*	{CF,HL} = {HL,CF}
RLA	17						fr		-	-	-	*	{CF,A} = {A,CF}
RLB A,BCDE	DD	6F							-	-	-	-	{A, BCDE} = {BCDE, A}
RLB A,JKHL	FD	6F							-	-	-	-	{A, JKHL} = {JKHL, A}
RLC (HL)	CB	06					f	b	*	*	P	*	(HL) = {(HL)[6,0],(HL)[7]}; CF = (HL)[7]
RLC (IX+d)	DD	CB	----d---	06			f	b	*	*	P	*	(IX+d) = {(IX+d)[6,0],(IX+d)[7]}; CF = (IX+d)[7]
RLC (IY+d)	FD	CB	----d---	06			f	b	*	*	P	*	(IY+d) = {(IY+d)[6,0],(IY+d)[7]}; CF = (IY+d)[7]
RLC r	CB	00000-r-					fr		*	*	P	*	r = {r[6,0],r[7]}; CF = r[7]
RLC 8,BCDE	DD	4F							-	-	-	-	BCDE = {CDE, B}
RLC 8,JKHL	FD	4F							-	-	-	-	JKHL = {KHL, J}
RLC bb,BCDE	DD	010010bb					fr		*	*	P	*	BCDE = {BCDE[30,0],B[7]}; CF = B[7]; bb=bb-1; repeat while bb!=0
RLC bb,JKHL	FD	010010bb					fr		*	*	P	*	JKHL = {JKHL[30,0],J[7]}; CF = J[7]; bb=bb-1; repeat while bb!=0
RLC BC	60						fr		*	*	P	*	BC = {BC[14,0],B[7]}; CF = B[7]
RLC DE	50						fr		*	*	P	*	DE = {DE[14,0],D[7]}; CF = D[7]
RLCA	07						fr		-	-	-	*	A = {A[6,0],A[7]}; CF = A[7]
RR (HL)	CB	1E					f	b	*	*	P	*	{(HL),CF} = {CF,(HL)}
RR (IX+d)	DD	CB	----d---	1E			f	b	*	*	P	*	{(IX+d),CF} = {CF,(IX+d)}
RR (IY+d)	FD	CB	----d---	1E			f	b	*	*	P	*	{(IY+d),CF} = {CF,(IY+d)}
RR BC	63						f		*	*	P	*	{BC,CF} = {CF,BC}
RR DE	FB						f		*	*	P	*	{DE,CF} = {CF,DE}
RR HL	FC						f		*	*	P	*	{HL,CF} = {CF,HL}
RR IX	DD	FC					f		*	*	P	*	{IX,CF} = {CF,IX}
RR IY	FD	FC					f		*	*	P	*	{IY,CF} = {CF,IY}
RR r	CB	00011-r-					fr		*	*	P	*	{r,CF} = {CF,r}
RR bb,BCDE	DD	011110bb					fr		*	*	P	*	{BCDE,CF} = {CF,BCDE}; bb=bb-1; repeat while bb!=0
RR bb,JKHL	FD	011110bb					fr		*	*	P	*	{JKHL,CF} = {CF,JKHL}; bb=bb-1; repeat while bb!=0
RRA	1F						fr		-	-	-	*	{A,CF} = {CF,A}

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Opcode byte 5	Opcode byte 6	AD	IO	S	Z	PV	C	Operation
RRB A,BCDE	DD	7F							-	-	-	-	{A, BCDE} = {E, A, BCD}
RRB A,JKHL	FD	7F							-	-	-	-	{A, JKHL} = {L, A, JKH}
RRC (HL)	CB	0E					f	b	*	*	P	*	(HL) = {(HL)[0],(HL)[7,1]}; CF = (HL)[0]
RRC (IX+d)	DD	CB	----d---	0E			f	b	*	*	P	*	(IX+d) = {(IX+d)[0],(IX+d)[7,1]}; CF = (IX+d)[0]
RRC (IY+d)	FD	CB	----d---	0E			f	b	*	*	P	*	(IY+d) = {(IY+d)[0],(IY+d)[7,1]}; CF = (IY+d)[0]
RRC r	CB	00001-r-					fr		*	*	P	*	r = {r[0],r[7,1]}; CF = r[0]
RRC 8,BCDE	DD	5F							-	-	-	-	BCDE = {E, BCD}
RRC 8,JKHL	FD	5F							-	-	-	-	JKHL = {L, JKH}
RRC bb,BCDE	DD	010110bb					fr		*	*	P	*	BCDE = {B[7],BCDE[31,1]}; CF = E[0]; bb=bb-1; repeat while bb!=0
RRC bb,JKHL	FD	010110bb					fr		*	*	P	*	JKHL = {J[7],JKHL[31,1]}; CF = L[0]; bb=bb-1; repeat while bb!=0
RRC BC	61						fr		*	*	P	*	BC = {B[0],BC[15,1]}; CF = C[0]
RRC DE	51						fr		*	*	P	*	DE = {D[0],DE[15,1]}; CF = E[0]
RRCA	0F						fr		-	-	-	*	A = {A[0],A[7,1]}; CF = A[0]
RST v	11-v-111								-	-	-	-	(SP-1) = PCH; (SP-2) = PCL; SP = SP - 2; PC = {R, 0, v, 0000}
SBC A,(HL)	9E						fr	s	*	*	V	*	A = A - (HL) - CF
SBC A,(HL)	7F	9E					fr	s	*	*	V	*	A = A - (HL) - CF
SBC A,n	DE	----n---					fr		*	*	V	*	A = A - n - CF
SBC A,r	10011-r-						fr		*	*	V	*	A = A - r - CF
SBC A,r	7F	10011-r-					fr		*	*	V	*	A = A - r - CF
SBC HL,ss	ED	01ss0010					fr		*	*	V	*	HL = HL - ss - CF
SBC (IX+d)	DD	9E	----d---				fr	s	*	*	V	*	A = A - (IX+d) - CF
SBC (IY+d)	FD	9E	----d---				fr	s	*	*	V	*	A = A - (IY+d) - CF
SBOX A	ED	02					r		-	-	-	-	A = sbox(A)
SCF	37						f		-	-	-	1	CF = 1
SET b,(HL)	CB	11-b-110						b	-	-	-	-	(HL) = (HL) | bit
SET b,(IX+d)	DD	CB	----d---	11-b-110				b	-	-	-	-	(IX+d) = (IX+d) | bit
SET b,(IY+d)	FD	CB	----d---	11-b-110				b	-	-	-	-	(IY+d) = (IY+d) | bit
SET b,r	CB	11-b--r-					r		-	-	-	-	r = r | bit
SETSYSP mn	ED	B1	n	m					-	-	-	-	SU={SU[1:0],SU[7:2]}; tmpl = (SP); tmph = (SP+1); SP = SP+2; if {tmp != mn} System Violation
SETUSR	ED	6F							-	-	-	-	SU = {SU[5:0], 01}
SETUSRP mn	ED	BF	n	m					-	-	-	-	SU = {SU[7:2], 01}; (SP - 1) = m; (SP - 2) = n; SP = SP - 2
SLA (HL)	CB	26					f	b	*	*	P	*	(HL) = {(HL)[6,0],0}; CF = (HL)[7]
SLA (IX+d)	DD	CB	----d---	26			f	b	*	*	P	*	(IX+d) = {(IX+d)[6,0],0}; CF = (IX+d)[7]

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Opcode byte 5	Opcode byte 6	AD	IO	S	Z	PV	C	Operation
SLA (IY+d)	FD	CB	----d---	26			f	b	*	*	P	*	(IY+d) = {(IY+d)[6,0],0}; CF = (IY+d)[7]
SLA r	CB	00100-r-					fr		*	*	P	*	r = {r[6,0],0}; CF = r[7]
SLA bb,BCDE	DD	100010bb					fr		*	*	P	*	BCDE = {BCDE[30,0],0}; CF = B[7]; bb=bb-1; repeat while bb!=0
SLA bb,JKHL	FD	100010bb					fr		*	*	P	*	JKHL = {JKHL[30,0],0}; CF = J[7]; bb=bb-1; repeat while bb!=0
SLL bb,BCDE	DD	101010bb					fr		*	*	P	*	BCDE = {BCDE[30,0],0}; CF = B[7]; bb=bb-1; repeat while bb!=0
SLL bb,JKHL	FD	101010bb					fr		*	*	P	*	JKHL = {JKHL[30,0],0}; CF = J[7]; bb=bb-1; repeat while bb!=0
SRA (HL)	CB	2E					f	b	*	*	P	*	(HL) = {(HL)[7],(HL)[7,1]}; CF = (HL)[0]
SRA (IX+d)	DD	CB	----d---	2E			f	b	*	*	P	*	(IX+d) = {(IX+d)[7],(IX+d)[7,1]}; CF = (IX+d)[0]
SRA (IY+d)	FD	CB	----d---	2E			f	b	*	*	P	*	(IY+d) = {(IY+d)[7],(IY+d)[7,1]}; CF = (IY+d)[0]
SRA r	CB	00101-r-					fr		*	*	P	*	r = {r[7],r[7,1]}; CF = r[0]
SRA bb,BCDE	DD	100110bb					fr		*	*	P	*	BCDE = {B[7],BCDE[31,1]}; CF = E[0]; bb=bb-1; repeat while bb!=0
SRA bb,JKHL	FD	100110bb					fr		*	*	P	*	JKHL = {J[7],JKHL[31,1]}; CF = L[0]; bb=bb-1; repeat while bb!=0
SYSRET	ED	83							-	-	-	-	SU = (SP); PCl = (SP+1); PCh = (SP+2); SP = SP+ 3
SRL bb,BCDE	DD	101110bb					fr		*	*	P	*	BCDE = {0,BCDE[31,1]}; CF = E[0]; bb=bb-1; repeat while bb!=0
SRL bb,JKHL	FD	101110bb					fr		*	*	P	*	JKHL = {0,JKHL[31,1]}; CF = L[0]; bb=bb-1; repeat while bb!=0
SRL (HL)	CB	3E					f	b	*	*	P	*	(HL) = {0,(HL)[7,1]}; CF = (HL)[0]
SRL (IX+d)	DD	CB	----d---	3E			f	b	*	*	P	*	(IX+d) = {0,(IX+d)[7,1]}; CF = (IX+d)[0]
SRL (IY+d)	FD	CB	----d---	3E			f	b	*	*	P	*	(IY+d) = {0,(IY+d)[7,1]}; CF = (IY+d)[0]
SRL r	CB	00111-r-					fr		*	*	P	*	r = {0,r[7,1]}; CF = r[0]
SUB (HL)	96						fr	s	*	*	V	*	A = A - (HL)
SUB (HL)	7F	96					fr	s	*	*	V	*	A = A - (HL)
SUB (IX+d)	DD	96	----d---				fr	s	*	*	V	*	A = A - (IX+d)
SUB (IY+d)	FD	96	----d---				fr	s	*	*	V	*	A = A - (IY+d)
SUB HL,DE	55						fr		-	-	-	*	HL = HL - DE
SUB HL,JK	45						fr		-	-	-	*	HL = HL - JK
SUB JKHL,BCDE	ED	D6					fr		-	-	-	*	JKHL = JKHL - BCDE
SUB n	D6	----n---					fr		*	*	V	*	A = A - n
SUB r	10010-r-						fr		*	*	V	*	A = A - r
SUB r	7F	10010-r-					fr		*	*	V	*	A = A - r
SURES	ED	7D							-	-	-	-	SU = {SU[1:0], SU[7:2]}
SYSCALL	ED	75							-	-	-	-	(SP-1) = PCH; (SP-2) = PCL; SP = SP - 2; PC = {R, 01100000}
TEST BC	ED	4C					f		*	*	P	0	BC | 0
TEST BCDE	DD	5C					f		*	*	P	0	BCDE | 0

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Opcode byte 5	Opcode byte 6	AD	IO	S	Z	PV	C	Operation
TEST HL	4C						f		*	*	P	0	HL | 0
TEST IX	DD	4C					f		*	*	P	0	IX | 0
TEST IY	FD	4C					f		*	*	P	0	IY | 0
TEST JKHL	FD	5C					f		*	*	P	0	JKHL | 0
UMA	ED	C0						-	-	-	-	*	{CF:DE':(HL)} = (IX) + [(IY) * DE + DE' + CF]; BC = BC-1; IX = IX+1; IY = IY+1; HL = HL+1; repeat while BC !=0
UMS	ED	C8						-	-	-	-	*	{CF:DE:(HL)} = (IX) - [(IY) * DE + DE' + CF]; BC = BC-1; IX = IX+1; IY = IY+1; HL = HL+1; repeat while BC !=0
XOR (HL)	AE						fr	s	*	*	P	0	A = [A & ~(HL)] | [~A & (HL)]
XOR (HL)	7F	AE					fr	s	*	*	P	0	A = [A & ~(HL)] | [~A & (HL)]
XOR HL,DE	54						fr		*	*	P	0	HL = HL ^ DE
XOR (IX+d)	DD	AE	----d---				fr	s	*	*	P	0	A = [A & ~(IX+d)] | [~A & (IX+d)]
XOR (IY+d)	FD	AE	----d---				fr	s	*	*	P	0	A = [A & ~(IY+d)] | [~A & (IY+d)]
XOR JKHL,BCDE	ED	EE					fr		*	*	P	0	JKHL = JKHL ^ BCDE
XOR n	EE	----n---					fr		*	*	P	0	A = [A & ~n] | [~A & n]
XOR r	10101-r-						fr		*	*	P	0	A = [A & ~r] | [~A & r]
XOR r	7F	10101-r-					fr		*	*	P	0	A = [A & ~r] | [~A & r]

Chapter 5. Opcode Map

This chapter lets you look up an instruction mnemonic by its opcode. There are two main pages, one for Rabbit 2000 and 3000 processors and one for the Rabbit 4000 and newer Rabbit processors. The main pages are followed by the prefix pages: ED, DD, FD, CB, DD-CB, FD-CB, 6D and 7F. Below are links to the various pages.

"Main Page, Rabbit 3000 Mode"	This is the main page for the Rabbit 2000 and Rabbit 3000. It is also the main page for the Rabbit 4000 and newer Rabbit processors when in Rabbit 3000 compatibility mode (i.e., the mode the processor boots up in.)
"Main Page, Rabbit 4000 Mode"	This is the main page is for the Rabbit 4000 and newer Rabbit processors; Rabbit 2000 and 3000 opcodes are not included.
"ED Page"	
"DD Page"	
"FD Page"	
"CB Page"	
"DD-CB Page"	
"FD-CB Page"	
"6D Page"	
"7F Page, Rabbit 4000 Mode"	

Main Page, Rabbit 3000 Mode

\LSB MSB\	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 label	LD DE, mn	LD (DE),A	INC DE	INC D	DEC D	LD D,n	RLA	JR label	ADD HL,DE	LD A, (DE)	DEC DE	INC E	DEC E	LD E,n	RRA
2	JR NZ, label	LD HL, mn	LD (mn),H L	INC HL	INC H	DEC H	LD H,n	ADD SP,d	JR Z, label	ADD HL,HL	LD HL, (mn)	DEC HL	INC L	DEC L	LD L,n	CPL
3	JR NC, label	LD SP, mn	LD (mn),A	INC SP	INC (HL)	DEC (HL)	LD (HL), n	SCF	JR C, label	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	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 lmn	RET Z	RET	JP Z,mn	PAGE CB	BOOL HL	CALL nn	ADC A,n	LCALL lmn
D	RET NC	POP DE	JP NC,mn	IOIP	LD (SP+n) , HL	PUSH DE	SUB n	RST 10	RET C	EXX	JP C,mn	IOEP	AND HL,DE	PAGE DD	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	PAGE ED	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	PAGE FD	CP n	RST 38

Main Page, Rabbit 4000 Mode

\LSB MSB\	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 label	ADD HL, DE	LD A,(DE)	DEC DE	INC E	DEC E	LD E,n	RRA
2	JR NZ, label	LD HL, mn	LD (mn),HL	INC HL	INC H	DEC H	LD H,n	ADD SP,d	JR Z, label	ADD HL, HL	LD HL,(mn)	DEC HL	INC L	DEC L	LD L,n	CPL
3	JR NC, label	LD SP, mn	LD (mn),A	INC SP	INC (HL)	DEC (HL)	LD (HL),n	SCF	JR C, label	ADD HL,SP	LD A,(mn)	DEC SP	INC A	DEC A	LD A,n	CCF
4			RL HL			SUB HL, JK	LD B,(HL)	LD B,A	CP HL,d				TEST HL	NEG HL	LD C,(HL)	LD C,A
5	RLC DE	RRC DE			XOR HL,DE	SUB HL, DE	LD D,(HL)	LD D,A				LD E,E			LD E,(HL)	LD E,A
6	RLC BC	RRC BC	RL BC	RR BC		ADD HL, JK	LD H,(HL)	LD H,A						PAGE 6D	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)	PAGE 7F
8		LD HL,BC	LDF (lmn), HL	LD (mn),BCDE	LD (mn), JKHL	LD HL,(PW+d)	LD (PW+d),HL	LLJP lxpc,mn		LD (mn),JK	LDF (lmn),A	LD A, (PW+HL)	LD (PW+HL), A	LD A,(PW+d)	LD (PW+d),A	LLCALL lxpc,mn
9		LD BC,HL	LDF HL, (lmn)	LD BCDE,(mn)	LD JKHL, (mn)	LD HL, (PX+d)	LD (PX+d),HL	LD XPC,HL	JRE label	LD JK, (mn)	LDF A,(lmn)	LD A, (PX+HL)	LD (PX+HL), A	LD A,(PX+d)	LD (PX+d),A	LD HL,XPC
A	JR GT, label	LD HL,DE	JP GT,mn	LD BCDE,d	LD JKHL,d	LD HL,(PY+d)	LD (PY+d),HL	MULU	JR GTU, label	LD JK, mn	JP GTU,mn	LD A, (PY+HL)	LD (PY+HL), A	LD A,(PY+d)	LD (PY+d),A	XOR A
B	JR LT, label	LD DE, HL	JP LT,mn	EX BC,HL	EX JKHL,BCDE	LD HL,(PZ+d)	LD (PZ+d),HL	OR A	JR V, label	EX JK, HL	JP V,mn	LD A,(PZ+HL)	LD (PZ+HL), A	LD A,(PZ+d)	LD (PZ+d),A	CLR HL
C	RET NZ	POP BC	JP NZ,mn	JP mn	LD HL, (SP+n)	PUSH BC	ADD A,n	LJP lmn	RET Z	RET	JP Z,mn	PAGE CB	BOOL HL	CALL nn	ADC A,n	LCALL lmn
D	RET NC	POP DE	JP NC,mn	IOIP	LD (SP+n), HL	PUSH DE	SUB n	RST 10	RET C	EXX	JP C,mn	IOEP	AND HL,DE	PAGE DD	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	PAGE ED	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	PAGE FD	CP n	RST 38

ED Page

\LSB MSB\	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	CBM n	LD PW,(HTR+HL)	SBOX A	LDL PW,(SP+n)	LD PW,(SP+n)	LD (SP+n),PW	LD HL,(PW+BC)	LD (PW+BC),HL	LDF PW,(lmn)	LDF (lmn),PW	LDF BC,(lmn)	LDF (lmn),BC	LD PW,klmn	LDL PW,mn	CONVC PW	CONVD PW
1	DWJNZ label	LD PX,(HTR+HL)	IBOX A	LDL PX,(SP+n)	LD PX,(SP+n)	LD (SP+n),PX	LD HL,(PX+BC)	LD (PX+BC),HL	LDF PX,(lmn)	LDF (lmn),PX	LDF DE,(lmn)	LDF (lmn),DE	LD PX,klmn	LDL PX,mn	CONVC PX	CONVD PX
2		LD PY,(HTR+HL)		LDL PY,(SP+n)	LD PY,(SP+n)	LD (SP+n),PY	LD HL,(PY+BC)	LD (PY+BC),HL	LDF PY,(lmn)	LDF (lmn),PY	LDF IX,(lmn)	LDF (lmn),IX	LD PY,klmn	LDL PY,mn	CONVC PY	CONVD PY
3		LD PZ,(HTR+HL)		LDL PZ,(SP+n)	LD PZ,(SP+n)	LD (SP+n),PZ	LD HL,(PZ+BC)	LD (PZ+BC),HL	LDF PZ,(lmn)	LDF (lmn),PZ	LDF IY,(lmn)	LDF (lmn),IY	LD PZ,klmn	LDL PZ,mn	CONVC PZ	CONVD PZ
4	LD HTR,A	LD BC',DE	SBC HL,BC	LD (mn),BC	NEG	LRET	IP 0	LD EIR,A	CP HL,DE	LD BC',BC	ADC HL,BC	LD BC,(mn)	TEST BC	RETI	IP 2	LD IIR,A
5	LD A,HTR	LD DE',DE	SBC HL,DE	LD (mn),DE	EX (SP),HL	SCALL	IP 1	LD A,EIR	CP JKHL,BCDE	LD DE',BC	ADC HL,DE	LD DE,(mn)		IPRET	IP 3	LD A,IIR
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	EX BC',HL	SYSCALL	PUSH IP	LD A,XPC			ADC HL,SP	LD SP,(mn)	EX JK',HL	SURES	POP IP	RDMODE
8	COPY			SRET					COPYR			LLRET				
9	LDISR								LDDSR							
A	LDI		LLJP GT,lxpc,mn	JRE GT,label	FLAG GT,HL	PUSH mn			LDD		LLJP GTU,lxpc,mn	JRE GTU,label	FLAG GTU,HL			
B	LDIR	SETSYSP mn	LLJP LT,lxpc,mn	JRE LT,label	FLAG LT,HL	SETUSRP mn			LDDR		LLJP V,lxpc,mn	JRE V,label	FLAG V,HL			
C	UMA	POP PW	LLJP NZ,lxpc,mn	JRE NZ,label	FLAG NZ,HL	PUSH PW	ADD JKHL,BCDE		UMS		LLJP Z,lxpc,mn	JRE Z,label	FLAG Z,HL			
D	LSIDR	POP PX	LLJP NC,lxpc,mn	JRE NC,label	FLAG NC,HL	PUSH PX	SUB JKHL,BCDE		LSDDR	EXP	LLJP C,lxpc,mn	JRE C,label	FLAG C,HL			
E		POP PY				PUSH PY	AND JKHL,BCDE				CALL (HL)				XOR JKHL,BCDE	
F	LSIR	POP PZ				PUSH PZ	OR JKHL,BCDE		LSDR		LLCALL (JKHL)				LD HL,(SP+HL)	

DD Page

\LSB MSB\	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0							LD A, (IX+A)			ADD IX,BC	LDF BCDE,(lmn)	LDF (lmn), BCDE	LD BCDE,(PW +H L)	LD (PW+HL),BCD E	LD BCDE,(P W+d)	LD (PW+d), BCDE
1										ADD IX,DE	LD BCDE,(HL)	LD (HL),B CDE	LD BCDE, (PX+HL)	LD (PX+HL),BCD E	LD BCDE,(P X+d)	LD (PX+d), BCDE
2		LD IX,mn	LD (mn),IX	INC IX						ADD IX,IX	LD IX,(mn)	DEC IX	LD BCDE,(PY +HL)	LD (PY+HL),BCD E	LD BCDE,(P Y+d)	LD (PY+d), BCDE
3					INC (IX+d)	DEC (IX+d)	LD (IX+d),n			ADD IX,SP			LD BCDE, (PZ+HL)	LD (PZ+HL),BCDE	LD BCDE,(P Z+d)	LD (PZ+d), BCDE
4							LD B, (IX+d)		RLC 1,BCDE	RLC 2, BCDE		RLC 4, BCDE	TEST IX	NEG BCDE	LD C,(IX+d)	RLC 8,BCDE
5							LD D, (IX+d)		RRC 1,BCDE	RRC 2, BCDE		RRC 4, BCDE	TEST BCDE		LD E,(IX+d)	RRC 8,BCDE
6					LDP (IX), HL	LDP (mn),IX	LD H, (IX+d)		RL 1,BCDE	RL 2, BCDE		RL 4, BCDE	LDP HL,(IX)	LDP IX,(mn)	LD L,(IX+d)	RLA 8,BCDE
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	RR 1,BCDE	RR 2, BCDE		RR 4, BCDE	LD HL,IX	LD IX,HL	LD A,(IX+d)	RRA 8,BCDE
8							ADD A, (IX+d)		SLA 1,BCDE	SLA 2, BCDE		SLA 4, BCDE	LDL PW,IX	LD PW,BCD E	ADC A,(IX+d)	LDL PW,DE
9							SUB (IX+d)		SRA 1,BCDE	SRA 2, BCDE		SRA 4, BCDE	LDL PX,IX	LD PX,BCD E	SBC A,(IX+d)	LDL PX,DE
A							AND (IX+d)		SLL 1,BCDE	SLL 2, BCDE		SLL 4, BCDE	LDL PY,IX	LD PY,BCD E	XOR (IX+d)	LDL PY,DE
B							OR (IX+d)		SRL 1,BCDE	SRL 2, BCDE		SRL 4, BCDE	LDL PZ,IX	LD PZ,BCD E	CP (IX+d)	LDL PZ,DE
C					LD IX, (SP+n)							PAGE DD-CB	BOOL IX	LD BCDE,P W	LD BCDE, (IX+d)	LD (IX+d), BCDE
D					LD (SP+n), IX								AND IX,DE	LD BCDE,P X	LD BCDE, (IY+d)	LD (IY+d), BCDE
E		POP IX		EX (SP),IX	LD HL, (HL+d)	PUSH IX				JP (IX)	CALL (IX)		OR IX,DE	LD BCDE,P Y	LD BCDE, (SP+n)	LD (SP+n), BCDE
F		POP BCDE			LD (HL+d), HL	PUSH BCDE				LD SP,IX			RR IX	LD BCDE,P Z	LD BCDE, (SP+HL)	LD (SP+HL) , BCDE

FD Page

\LSB MSB\	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0							LD A, (IY+A)			ADD IY,BC	LDF JKHL,(lmn)	LDF (lmn), JKHL	LD JKHL,(PW+HL)	LD (PW+HL) ,JKHL	LD JKHL, (PW+d)	LD (PW+d), JKHL
1										ADD IY,DE	LD JKHL,(HL)	LD (HL),J KHL	LD JKHL, (PX+HL)	LD (PX+HL) , JKHL	LD JKHL,(P X+d)	LD (PX+d), JKHL
2		LD IY,mn	LD (mn),IY	INC IY						ADD IY,IY	LD IY,(mn)	DEC IY	LD JKHL, (PY+HL)	LD (PY+HL) ,JKHL	LD JKHL, (PY+d)	LD (PY+d), JKHL
3					INC (IY+d)	DEC (IY+d)	LD (IY+d), n			ADD IX,SP			LD JKHL, (PZ+HL)	LD (PZ+HL) ,JKHL	LD JKHL, (PZ+d)	LD (PZ+d), JKHL
4							LD B, (IY+d)		RLC 1, JKHL	RLC 2,JKHL		RLC 4, JKHL	TEST IY	NEG JKHL	LD C, (IY+d)	RLC 8,JKHL
5							LD D, (IY+d)		RRC 1, JKHL	RRC 2, JKHL		RRC 4, JKHL	TEST JKHL		LD E, (IY+d)	RRC 8,JKHL
6					LDP (IY),HL	LDP (mn),IY	LD H, (IY+d)		RL 1, JKHL	RL 2, JKHL		RL 4,JKHL	LDP HL,(IY)	LDP IY,(mn)	LD L, (IY+d)	RLA 8,JKHL
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	RR 1, JKHL	RR 2, JKHL		RR 4, JKHL	LD HL,IY	LD IY,HL	LD A, (IY+d)	RRA 8,JKHL
8							ADD A, (IY+d)		SLA 1, JKHL	SLA 2, JKHL		SLA 4, JKHL	LDL PW,IY	LD PW,JKHL	ADC A, (IY+d)	LDL PW,HL
9							SUB (IY+d)		SRA 1, JKHL	SRA 2, JKHL		SRA 4, JKHL	LDL PX,IY	LD PX,JKHL	SBC A, (IY+d)	LDL PX,HL
A							AND (IY+d)		SLL 1, JKHL	SLL 2, JKHL		SLL 4, JKHL	LDL PY,IY	LD PY,JKHL	XOR (IY+d)	LDL PY,HL
B							OR (IY+d)		SRL 1, JKHL	SRL 2, JKHL		SRL 4, JKHL	LDL PZ,IY	LD PZ,JKHL	CP (IY+d)	LDL PZ,HL
C					LD IY, (SP+n)							PAGE FD-CB	BOOL IY	LD JKHL,PW	LD JKHL, (IX+d)	LD (IX+d), JKHL
D					LD (SP+n), IY								AND IY,DE	LD JKHL,PX	LD JKHL, (IY+d)	LD (IY+d), JKHL
E		POP IY		EX (SP),IY	LD HL, (IY+d)	PUSH IY				JP (IY)	CALL (IY)		OR IY,DE	LD JKHL,PY	LD JKHL, (SP+n)	LD (SP+n), JKHL
F		POP JKHL			LD (IY+d), HL	PUSH JKHL				LD SP,IY			RR IY	LD JKHL,PZ	LD JKHL, (SP+HL)	LD (SP+HL), JKHL

CB Page

\LSB MSB\	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	4,C BIT	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	6,D RES	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

DD-CB Page

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

FD-CB Page

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

6D Page

\LSB MSB\	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	LD BC,(PW+d)	LD (PW+d),BC	LD BC,(PW+HL)	LD (PW+HL),BC	LD PW,PW+IX	LD PW,PW+IY	LD PW,PW+DE	LD PW,PW	LD PW,(PW+d)	LD (PW+d),PW	LD PW,(PW+HL)	LD (PW+HL),PW	LD PW,PW+d		LD PW,PW+HL	
1	LD BC,(PX+d)	LD (PX+d),BC	LD BC,(PX+HL)	LD (PX+HL),BC	LD PW,PX+IX	LD PW,PX+IY	LD PW,PX+DE	LD PW,PX	LD PW,(PX+d)	LD (PX+d),PW	LD PW,(PX+HL)	LD (PX+HL),PW	LD PW,PX+d		LD PW,PX+HL	
2	LD BC,(PY+d)	LD (PY+d),BC	LD BC,(PY+HL)	LD (PY+HL),BC	LD PW,PY+IX	LD PW,PY+IY	LD PW,PY+DE	LD PW,PY	LD PW,(PY+d)	LD (PY+d),PW	LD PW,(PY+HL)	LD (PY+HL),PW	LD PW,PY+d		LD PW,PY+HL	
3	LD BC,(PZ+d)	LD (PZ+d),BC	LD BC,(PZ+HL)	LD (PZ+HL),BC	LD PW,PZ+IX	LD PW,PZ+IY	LD PW,PZ+DE	LD PW,PZ	LD PW,(PZ+d)	LD (PZ+d),PW	LD PW,(PZ+HL)	LD (PZ+HL),PW	LD PW,PZ+d		LD PW,PZ+HL	
4	LD DE,(PW+d)	LD (PW+d),DE	LD DE,(PW+HL)	LD (PW+HL),DE	LD PX,PW+IX	LD PX,PW+IY	LD PX,PW+DE	LD PX,PW	LD PX,(PW+d)	LD (PW+d),PX	LD PX,(PW+HL)	LD (PW+HL),PX	LD PX,PW+d		LD PX,PW+HL	
5	LD DE,(PX+d)	LD (PX+d),DE	LD DE,(PX+HL)	LD (PX+HL),DE	LD PX,PX+IX	LD PX,PX+IY	LD PX,PX+DE	LD PX,PX	LD PX,(PX+d)	LD (PX+d),PX	LD PX,(PX+HL)	LD (PX+HL),PX	LD PX,PX+d		LD PX,PX+HL	
6	LD DE,(PY+d)	LD (PY+d),DE	LD DE,(PY+HL)	LD (PY+HL),DE	LD PX,PY+IX	LD PX,PY+IY	LD PX,PY+DE	LD PX,PY	LD PX,(PY+d)	LD (PY+d),PX	LD PX,(PY+HL)	LD (PY+HL),PX	LD PX,PY+d	LD L,L	LD PX,PY+HL	
7	LD DE,(PZ+d)	LD (PZ+d),DE	LD DE,(PZ+HL)	LD (PZ+HL),DE	LD PX,PZ+IX	LD PX,PZ+IY	LD PX,PZ+DE	LD PX,PZ	LD PX,(PZ+d)	LD (PZ+d),PX	LD PX,(PZ+HL)	LD (PZ+HL),PX	LD PX,PZ+d		LD PX,PZ+HL	LD A,A
8	LD IX,(PW+d)	LD (PW+d),IX	LD IX,(PW+HL)	LD (PW+HL),IX	LD PY,PW+IX	LD PY,PW+IY	LD PY,PW+DE	LD PY,PW	LD PY,(PW+d)	LD (PW+d),PY	LD PY,(PW+HL)	LD (PW+HL),PY	LD PY,PW+d		LD PY,PW+HL	
9	LD IX,(PX+d)	LD (PX+d),IX	LD IX,(PX+HL)	LD (PX+HL),IX	LD PY,PX+IX	LD PY,PX+IY	LD PY,PX+DE	LD PY,PX	LD PY,(PX+d)	LD (PX+d),PY	LD PY,(PX+HL)	LD (PX+HL),PY	LD PY,PX+d		LD PY,PX+HL	
A	LD IX,(PY+d)	LD (PY+d),IX	LD IX,(PY+HL)	LD (PY+HL),IX	LD PY,PY+IX	LD PY,PY+IY	LD PY,PY+DE	LD PY,PY	LD PY,(PY+d)	LD (PY+d),PY	LD PY,(PY+HL)	LD (PY+HL),PY	LD PY,PY+d		LD PY,PY+HL	
B	LD IX,(PZ+d)	LD (PZ+d),IX	LD IX,(PZ+HL)	LD (PZ+HL),IX	LD PY,PZ+IX	LD PY,PZ+IY	LD PY,PZ+DE	LD PY,PZ	LD PY,(PZ+d)	LD (PZ+d),PY	LD PY,(PZ+HL)	LD (PZ+HL),PY	LD PY,PZ+d		LD PY,PZ+HL	
C	LD IY,(PW+d)	LD (PW+d),IY	LD IY,(PW+HL)	LD (PW+HL),IY	LD PZ,PW+IX	LD PZ,PW+IY	LD PZ,PW+DE	LD PZ,PW	LD PZ,(PW+d)	LD (PW+d),PZ	LD PZ,(PW+HL)	LD (PW+HL),PZ	LD PZ,PW+d		LD PZ,PW+HL	
D	LD IY,(PX+d)	LD (PX+d),IY	LD IY,(PX+HL)	LD (PX+HL),IY	LD PZ,PX+IX	LD PZ,PX+IY	LD PZ,PX+DE	LD PZ,PX	LD PZ,(PX+d)	LD (PX+d),PZ	LD PZ,(PX+HL)	LD (PX+HL),PZ	LD PZ,PX+d		LD PZ,PX+HL	
E	LD IY,(PY+d)	LD (PY+d),IY	LD IY,(PY+HL)	LD (PY+HL),IY	LD PZ,PY+IX	LD PZ,PY+IY	LD PZ,PY+DE	LD PZ,PY	LD PZ,(PY+d)	LD (PY+d),PZ	LD PZ,(PY+HL)	LD (PY+HL),PZ	LD PZ,PY+d		LD PZ,PY+HL	
F	LD IY,(PZ+d)	LD (PZ+d),IY	LD IY,(PZ+HL)	LD (PZ+HL),IY	LD PZ,PZ+IX	LD PZ,PZ+IY	LD PZ,PZ+DE	LD PZ,PZ	LD PZ,(PZ+d)	LD (PZ+d),PZ	LD PZ,(PZ+HL)	LD (PZ+HL),PZ	LD PZ,PZ+d		LD PZ,PZ+HL	

7F Page, Rabbit 4000 Mode

\LSB MSB\	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2																
3																
4	LD B,B	LD B,C	LD B,D	LD B,E	LD B,H	LD B,L		LD B,A	LD C,B	LD C,C	LD C,D	LD C,E	LD C,H	LD C,L		LD C,A
5	LD D,B	LD D,C	LD D,D	LD D,E	LD D,H	LD D,L		LD D,A	LD E,B	LD E,C	LD E,D	LD E,E	LD E,H	LD E,L		LD E,A
6	LD H,B	LD H,C	LD H,D	LD H,E	LD H,H	LD H,L		LD H,A	LD L,B	LD L,C	LD L,D	LD L,E	LD L,H	LD L,L		LD L,A
7									LD A,B	LD A,C	LD A,D	LD A,E	LD A,H	LD A,L		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																
D																
E																
F																