

The Undocumented Z80 Documented

Sean Young

Version 0.4, September 2001

Copyright Statement

Copyright ©1997, 1998, 2001 Sean Young.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 1.1 | History | 1 |
| 1.2 | Where to get this document | 1 |
| 1.3 | Feedback | 1 |
| 2 | Overview | 2 |
| 2.1 | History of the Z80 | 2 |
| 2.2 | Registers | 2 |
| 2.3 | Flags | 3 |
| 2.4 | Power on defaults | 4 |
| 2.5 | Pin Descriptions [7] | 4 |
| 3 | Undocumented Opcodes | 7 |
| 3.1 | CB Prefix [5] | 7 |
| 3.2 | DD Prefix [5] | 7 |
| 3.3 | FD Prefix [5] | 8 |
| 3.4 | ED Prefix [5] | 8 |
| 3.5 | DDCB Prefix | 9 |
| 3.6 | FDCB Prefixes | 11 |
| 3.7 | Combinations of Prefixes | 11 |
| 4 | Undocumented Instructions | 12 |
| 4.1 | BIT instructions | 12 |
| 4.2 | Memory Block Instructions [1] | 13 |
| 4.3 | I/O Block Instructions | 13 |
| 4.4 | 16 bit I/O ports | 14 |
| 4.5 | Block Instructions | 14 |
| 4.6 | 16 Bit Additions | 14 |
| 4.7 | DAA Instruction | 14 |
| 5 | Interrupts | 16 |
| 5.1 | Non-Maskable Interrupts (NMI) | 16 |
| 5.2 | Maskable Interrupts (INT) | 16 |
| 5.3 | Things affecting the Interrupt flip-flops | 17 |
| 5.4 | HALT instruction | 18 |
| 5.5 | Where interrupts are accepted | 18 |

| | | |
|-----------|--|-----------|
| 6 | Timing and R register | 20 |
| 6.1 | R register and memory refresh | 20 |
| 6.2 | Undocumented Timings | 20 |
| 7 | Other Information | 22 |
| 7.1 | Errors in official documentation | 22 |
| 8 | Instruction Tables | 24 |
| 8.1 | 8-Bit Load Group | 24 |
| 8.2 | 16-Bit Load Group | 25 |
| 8.3 | Exchange, Block Transfer, Block Search Group | 27 |
| 8.4 | 8-Bit Arithmetic and Logical Group | 28 |
| 8.5 | General-Purpose Arithmetic and CPU Control Group | 28 |
| 8.6 | 16-Bit Arithmetic Group | 29 |
| 8.7 | Rotate and Shift Group | 30 |
| 8.8 | Bit Set, Reset and Test Group | 31 |
| 8.9 | Jump Group | 31 |
| 8.10 | Call and Return Group | 32 |
| 8.11 | Input and Output Group | 33 |
| 9 | Instructions Sorted by Opcode | 34 |
| 10 | GNU Free Documentation License | 41 |
| 10.1 | Applicability and Definitions | 41 |
| 10.2 | Verbatim Copying | 42 |
| 10.3 | Copying in Quantity | 43 |
| 10.4 | Modifications | 43 |
| 10.5 | Combining Documents | 45 |
| 10.6 | Collections of Documents | 45 |
| 10.7 | Aggregation With Independent Works | 46 |
| 10.8 | Translation | 46 |
| 10.9 | Termination | 46 |
| 10.10 | Future Revisions of This License | 46 |

Chapter 1

Introduction

1.1 History

Ever since I first started writing an MSX emulator, I've been very interested in getting the emulation absolutely correct — including the undocumented features. Not only to make sure that all games work, but also to make sure that if a program crashes, it crashes exactly the same way if running on an emulator as on the real thing. Only then is perfection achieved.

I set about collecting information. I found pieces of information on the Internet, but not everything there is to know. So I tried to fill in the gaps, the results of which I put on my website. Various people have helped me since then; this is the result of all those efforts and to my knowledge this document covers more undocumented features than any other.

Previous documents I had written were in plain text and Microsoft Word, which I now find very embarrassing, so I decided to combine them all and use \LaTeX . Apart from a full re-write, the only changed information is “Power on defaults” (section 2.4) and the algorithm for the CF and HF flags for OTIR and friends (section 4.3).

1.2 Where to get this document

The latest version is always available in \LaTeX and pdf at the following location:
<http://www.msxnet.org/tech/>

1.3 Feedback

I welcome any kind of feedback. I would like to hear about any mistakes in this document or any additions you might have. Also note that there are a few flags which are still unknown, I would be most grateful if someone could figure them out (please include an instruction exerciser to show that your findings are correct).

You can reach me at sean@msxnet.org and my website can be found at <http://www.msxnet.org/>.

Chapter 2

Overview

2.1 History of the Z80

In 1969 Intel were approached by a Japanese company called Busicom to produce chips for Busicom's electronic desktop calculator. Intel suggested that the calculator should be built around a single-chip generalized computing engine and thus was born the first microprocessor — the 4004. Although it was based on ideas from much larger mainframe and mini-computers the 4004 was cut down to fit onto a 16-pin chip, the largest that was available at the time, so that its data bus and address bus were each only 4-bits wide.

Intel went on to improve the design and produced the 4040 (an improved 4-bit design) the 8008 (the first 8-bit microprocessor) and then in 1974 the 8080. This last one turned out to be a very useful and popular design and was used in the first home computer, the Altair 8800, and CP/M.

In 1975 Federico Faggin who had had worked at Intel on the 4004 and its successors left the company and joined forces with Masatoshi Shima to form Zilog. At their new company Faggin and Shima designed a microprocessor that was compatible with Intel's 8080 (it ran all 78 instructions of the 8080 in exactly the same way that Intel's chip did) but had many more abilities (an extra 120 instructions, many more registers, simplified connection to hardware). Thus was born the mighty Z80!

The original Z80 was first released in July 1976. Since then newer versions have appeared with exactly the same architecture but running at higher speeds. The original Z80 ran with a clock rate of 2.5MHz, the Z80A runs at 4MHz, the Z80B at 6MHz, and the Z80H at 8Mhz.

Many companies produced machines based around Zilog's improved chip during the 1970's and 80's and because the chip could run 8080 code without needing any changes to the code the perfect choice of operating system was CP/M.

2.2 Registers

The following accessible registers exist in the Z80.

| | | |
|-----|---|-------------------------------------|
| A | F | Accumulator and Flags |
| BC | | |
| DE | | General purpose registers |
| HL | | |
| IX | | |
| IY | | Index registers |
| PC | | |
| SP | | Special purpose registers |
| I | R | |
| AF' | | Alternate general purpose registers |
| BC' | | |
| DE' | | |
| HL' | | |

For interrupts, there are two interrupt flop-flops, IFF1 and IFF2, and the interrupt mode is remembered. See chapter 5. Also there is an internal register which is described in section 4.3.

2.3 Flags

The conventional way of denoting the flags is with one letter, 'C' for the carry flag for example. It could be confused with the C register, so I've chosen to use the 'CF' notation for flags. Also in previous things I've written I called the two undocumented flags 5 and 3, but now I've changed to the same notation used in MAME¹, which is YF and XF, respectively.

| | | | | | | | | |
|------|----|----|----|----|----|----|----|----|
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| flag | SF | ZF | YF | HF | XF | PF | NF | CF |

SF flag Set if the 2-complement value is negative. It's simply a copy of the most significant bit.

ZF flag Set if the value is zero.

YF flag A copy of bit 5 of the result.

HF flag The half-carry of an addition/subtraction (from bit 3 to 4). Needed for BCD correction with DAA.

XF flag A copy of bit 3 of the result.

PF flag This flag can either be the parity of the result (PF), or the 2-compliment signed overflow (VF): set if 2-compliment value doesn't fit in the register.

NF flag Shows whether the last operation was an addition (0) or an subtraction (1). This information is needed for DAA.

CF flag The carry flag, set if there was a carry after the most significant bit.

¹<http://www.mame.net/>

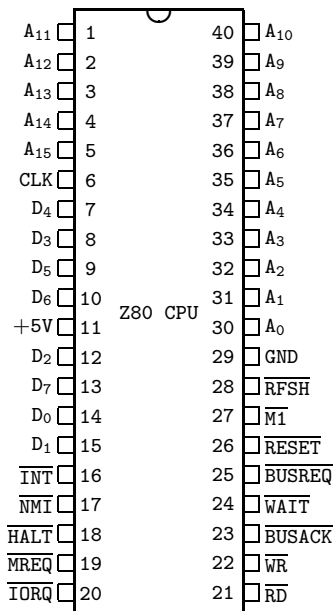
2.4 Power on defaults

Matt² has done some excellent research for this. He found that AF and SP are always set to FFFFh after a reset, and all other registers are undefined (different depending on how long the CPU has been powered off, different for different Z80 chips). Of course the PC should be set to 0 after a reset, and so should the IFF1 and IFF2 flags (otherwise strange things could happen). Also since the Z80 is 8080 compatible, interrupt mode is probably 0.

Probably the best way to simulate this in an emulator is set PC, IFF1, IFF2, IM to 0 and set all other registers to FFFFh.

2.5 Pin Descriptions [7]

This section is also relevant even if you don't do anything with hardware. Besides, it took me hours to draw this picture.



A₁₅ – A₀ *Address bus* (output, active high, 3-state). This bus is used for accessing the memory and for I/O ports. During the refresh cycle the IR register is put on this bus.

$\overline{\text{BUSACK}}$ *Bus Acknowledge* (output, active low). Bus Acknowledge indicates to the requesting device that the CPU address bus, data bus, and control signals $\overline{\text{MREQ}}$, $\overline{\text{IORQ}}$, $\overline{\text{RD}}$ and $\overline{\text{WR}}$ have been entered into their high-impedance states. The external now control these lines.

$\overline{\text{BUSREQ}}$ *Bus Request* (input, active low). Bus Request has a higher priority than $\overline{\text{NMI}}$ and is always recognised at the end of the current machine cycle.

$\overline{\text{BUSREQ}}$ forces the CPU address bus, data bus and control signals $\overline{\text{MREQ}}$,

²redflame@xmission.com

$\overline{\text{TORQ}}$, $\overline{\text{RD}}$ and $\overline{\text{WR}}$ to go to a high-impedance state so that other devices can control these lines. $\overline{\text{BUSREQ}}$ is normally wired-OR and requires an external pullup for these applications. Extended $\overline{\text{BUSREQ}}$ periods due to extensive DMA operations can prevent the CPU from refreshing dynamic RAMs.

$\text{D}_7 - \text{D}_0$ *Data Bus* (input/output, active low, 3-state). Used for data exchanges with memory, I/O and interrupts.

$\overline{\text{HALT}}$ *Halt State* (output, active low). Indicates that the CPU has executed a HALT instruction and is waiting for either a maskable or nonmaskable interrupt (with the mask enabled) before operation can resume. While halted, the CPU stops increasing the PC so the instruction is re-executed, to maintain memory refresh.

$\overline{\text{INT}}$ *Interrupt Request* (input, active low). Interrupt Request is generated by I/O devices. The CPU honours a request at the end of the current instruction if IFF1 is set. $\overline{\text{INT}}$ is normally wired-OR and requires an external pullup for these applications.

$\overline{\text{IORQ}}$ *Input/Output Request* (output, active low, 3-state). Indicates that the address bus holds a valid I/O address for an I/O read or write operation. $\overline{\text{IORQ}}$ is also generated concurrently with $\overline{\text{M1}}$ during an interrupt acknowledge cycle to indicate that an interrupt response vector can be placed on the databus.

$\overline{\text{M1}}$ *Machine Cycle One* (output, active low). $\overline{\text{M1}}$, together with $\overline{\text{MREQ}}$, indicates that the current machine cycle is the opcode fetch cycle of an instruction execution. $\overline{\text{M1}}$, together with $\overline{\text{IORQ}}$, indicates an interrupt acknowledge cycle.

$\overline{\text{MREQ}}$ *Memory Request* (output, active low, 3-state). Indicates that the address holds a valid address for a memory read or write cycle operations.

$\overline{\text{NMI}}$ *Non-Maskable Interrupt* (input, negative edge-triggered). $\overline{\text{NMI}}$ has a higher priority than $\overline{\text{INT}}$. $\overline{\text{NMI}}$ is always recognised at the end of the current instruction, independent of the status of the interrupt flip-flops and automatically forces the CPU to restart at location 0066h.

$\overline{\text{RD}}$ *Read* (output, active low, 3-state). Indicates that the CPU wants to read data from memory or an I/O device. The addressed I/O device or memory should use this signal to place data onto the data bus.

$\overline{\text{RESET}}$ *Reset* (input, active low). Initializes the CPU as follows: it resets the interrupt flip-flops, clears the PC and IR registers, and set the interrupt mode to 0. During reset time, the address bus and data bus go to a high-impedance state, and all control output signals go to the inactive state. Note that $\overline{\text{RESET}}$ must be active for a minimum of three full clock cycles before the reset operation is complete. Note that Matt found that SP and AF are set to FFFFh.

$\overline{\text{RFSH}}$ *Refresh* (output, active low). $\overline{\text{RFSH}}$, together with $\overline{\text{MREQ}}$, indicates that the IR registers are on the address bus (note that only the lower 7 bits are useful) and can be used for the refresh of dynamic memories.

$\overline{\text{WAIT}}$ *Wait* (input, active low). Indicates to the CPU that the addressed memory or I/O device are not ready for data transfer. The CPU continues to enter a wait state as long as this signal is active. Note that during this period memory is not refreshed.

$\overline{\text{WR}}$ *Write* (output, active low, 3-state). Indicates that the CPU wants to write data to memory or an I/O device. The addressed I/O device or memory should use this signal to store the data on the data bus.

Chapter 3

Undocumented Opcodes

There are quite a few undocumented opcodes/instructions. This section should describe every possible opcode (so you know what will be executed, whatever the value of the opcode is).

The following prefixes exist: CB, ED, DD, FD, DDCB and FDCB. Prefixes change the way the following opcodes are interpreted. All instructions without a prefix (not a value of one the above) are single byte opcodes, which are documented in the official documentation.

3.1 CB Prefix [5]

An opcode with a CB prefix is a rotate, shift or bit test/set/reset instruction. There are a few instructions missing from the official list, which are usually denoted with SLL (Shift Logical Left). It works like SLA, for one exception: it sets bit 0 (SLA resets it).

| | |
|------|----------|
| CB30 | SLL B |
| CB31 | SLL C |
| CB32 | SLL D |
| CB33 | SLL E |
| CB34 | SLL H |
| CB35 | SLL L |
| CB36 | SLL (HL) |
| CB37 | SLL A |

3.2 DD Prefix [5]

In general, the instruction following the DD prefix is executed as is, but if the HL register is supposed to be used the IX register is used instead. Here are the rules:

- Any usage of HL is treated as an access to IX (except EX DE,HL and EXX and the ED prefixed instructions that use HL).

- Any access to (HL) is changed to (IX+d), where ‘d’ is a signed displacement byte placed after the main opcode — except JP (HL), which isn’t indirect anyway. The mnemonic should be JP HL.
- Any access to H is treated as an access to IXh (the high byte of IX) Except if (IX+d) is used as well.
- Any access to L is treated as an access to IXl (the low byte of IX) Except if (IX+d) is used aswell.
- A DD prefix before a CB selects a completely different instruction set, see Section 3.5.

Some examples:

| Without DD prefix | With DD prefix |
|-------------------|----------------|
| LD H, (HL) | LD H, (IX+d) |
| LD H, A | LD IXh, A |
| LD L, H | LD IXl, IXh |
| JP (HL) | JP (IX) |
| LD DE, 0 | LD DE, 0 |
| LD HL, 0 | LD IX, 0 |

3.3 FD Prefix [5]

This prefix has the same effect as the DD prefix, though IY is used instead of IX. Note LD IXl, IYh is not possible: only IX or IY is accessed in one instruction.

3.4 ED Prefix [5]

There are a number of undocumented EDxx instructions, of which most are duplicates of documented instructions. Any instruction not listed has no effect (same behaviour as 2 NOP instructions).

The complete list except for the block instructions:

| | |
|------------------|------------------|
| ED40 IN B, (C) | ED60 IN H, (C) |
| ED41 OUT (C), B | ED61 OUT (C), H |
| ED42 SBC HL, BC | ED62 SBC HL, HL |
| ED43 LD (nn), BC | ED63 LD (nn), HL |
| ED44 NEG | ED64 NEG** |
| ED45 RETN | ED65 RETN** |
| ED46 IM 0 | ED66 IM 0** |
| ED47 LD I, A | ED67 RRD |
| ED48 IN C, (C) | ED68 IN L, (C) |
| ED49 OUT (C), C | ED69 OUT (C), L |
| ED4A ADC HL, BC | ED6A ADC HL, HL |
| ED4B LD BC, (nn) | ED6B LD HL, (nn) |

**Undocumented instruction

| | | | |
|------|------------|------|---------------------|
| ED4C | NEG** | ED6C | NEG** |
| ED4D | RETI | ED6D | RETN** |
| ED4E | IM 0** | ED6E | IM 0** |
| ED4F | LD R,A | ED6F | RLD |
| ED50 | IN D,(C) | ED70 | IN (C) / IN F,(C)** |
| ED51 | OUT (C),D | ED71 | OUT (C),0** |
| ED52 | SBC HL,DE | ED72 | SBC HL,SP |
| ED53 | LD (nn),DE | ED73 | LD (nn),SP |
| ED54 | NEG** | ED74 | NEG** |
| ED55 | RETN** | ED75 | RETN** |
| ED56 | IM 1 | ED76 | IM 1** |
| ED57 | LD A,I | ED77 | NOP** |
| ED58 | IN E,(C) | ED78 | IN A,(C) |
| ED59 | OUT (C),E | ED79 | OUT (C),A |
| ED5A | ADC HL,DE | ED7A | ADC HL,SP |
| ED5B | LD DE,(nn) | ED7B | LD SP,(nn) |
| ED5C | NEG** | ED7C | NEG** |
| ED5D | RETN** | ED7D | RETN** |
| ED5E | IM 2 | ED7E | IM 2** |
| ED5F | LD A,R | ED7F | NOP** |

The ED70 instruction reads from I/O port C, but does not store the result. It just affects the flags like the other IN x,(C) instruction. ED71 simply outs the value 0 to I/O port C.

The ED63 is a duplicate of the 22 opcode (LD (nn),HL) and similarly ED6B is a duplicate of the 2A opcode. Of course the timings are different. These instructions are listed in the official documentation.

According to Gerton Lunter¹:

The instructions ED 4E and ED 6E are IM 0 equivalents: when FF was put on the bus (physically) at interrupt time, the Spectrum continued to execute normally, whereas when an EF (RST 28h) was put on the bus it crashed, just as it does in that case when the Z80 is in the official interrupt mode 0. In IM 1 the Z80 just executes a RST 38h (opcode FF) no matter what is on the bus.

All the RETI/RETN instructions are the same, all like the RETN instruction. So they all, including RETI, copy IFF2 to IFF1. More information on RETI and RETN and IM x is in section 5.3.

3.5 DDCB Prefix

The undocumented DDCB instructions store the result (if any) of the operation in one of the seven all-purpose registers, which one depends on the lower 3 bits of the last byte of the opcode (not operand, so not the offset).

¹gerton@math.rug.nl

```

000 B
001 C
010 D
011 E
100 H
101 L
110 (none: documented opcode)
111 A

```

The documented DDCB0106 is `RLC (IX+01h)`. So, clear the lower three bits (DDCB0100) and something is done to register B. The result of the RLC (which is stored in (IX+01h)) is now also stored in register B. Effectively, it does the following:

```

LD B, (IX+01h)
RLC B
LD (IX+01h), B

```

So you get double value for money. The result is stored in B and (IX+01h). The most common notation is: `RLC (IX+01h), B`

I've once seen this notation:

```

RLC (IX+01h)
LD B, (IX+01h)

```

That's not correct: B contains the rotated value, even if (IX+01h) points to ROM. The DDCB SET and RES instructions do the same thing as the shift/rotate instructions:

```

DDCB10C0    SET 0, (IX+10h), B
DDCB10C1    SET 0, (IX+10h), C
DDCB10C2    SET 0, (IX+10h), D
DDCB10C3    SET 0, (IX+10h), E
DDCB10C4    SET 0, (IX+10h), H
DDCB10C5    SET 0, (IX+10h), L
DDCB10C6    SET 0, (IX+10h) - documented instruction
DDCB10C7    SET 0, (IX+10h), A

```

So for example with the last instruction, the value of (IX+10h) with bit 0 set is also stored in register A.

The DDCB BIT instructions do not store any value; they merely test a bit. That's why the undocumented DDCB BIT instructions are no different from the official ones:

```

DDCB d 78   BIT 7, (IX+d)
DDCB d 79   BIT 7, (IX+d)
DDCB d 7A   BIT 7, (IX+d)
DDCB d 7B   BIT 7, (IX+d)
DDCB d 7C   BIT 7, (IX+d)
DDCB d 7D   BIT 7, (IX+d)
DDCB d 7E   BIT 7, (IX+d) - documented instruction
DDCB d 7F   BIT 7, (IX+d)

```

3.6 FDCB Prefixes

Same as for the DDCB prefix, though IY is used instead of IX.

3.7 Combinations of Prefixes

This part may be of some interest to emulator coders. Here we define what happens if strange sequences of prefixes appear in the instruction cycle of the Z80.

If CB or ED is encountered, that byte plus the next make up an instruction. FD or DD should be seen as prefix setting a flag which says “use IX or IY instead of HL”, and not an instruction. In a large sequence of DD and FD bytes, it is the last one that counts. Also any other byte (or instruction) resets this flag.

```
FD DD 00 21 00 10    NOP NOP NOP LD HL,1000h
```

Chapter 4

Undocumented Instructions

4.1 BIT instructions

BIT n,r behaves much like AND $r,2^n$ with the result thrown away, and CF flag unaffected. So compare BIT 7,A with AND 80h: flag YF and XF are reset, SF is set if bit 7 was actually set; ZF is set if the result was 0 (bit was reset), and PF is effectively set if ZF is set (the result of the AND leaves either no bits set (PF reset) or one bit set (PF set)). So the rules for the flags are:

SF flag Set if $n = 7$ and tested bit is set.

ZF flag Set if the tested bit is reset.

YF flag Set if $n = 5$ and tested bit is set.

HF flag Always set.

XF flag Set if $n = 3$ and tested bit is set.

PF flag Set just like ZF flag.

NF flag Always reset.

CF flag Unchanged.

This is where things start to get strange. With the BIT $n,(IX+d)$ instructions, the flags behave just like the BIT n,r instruction, except for YF and XF. These are not copied from the result but from something completely different, namely bit 5 and 3 of the high byte of IX+d (so IX plus the displacement).

Things get more bizarre with the BIT $n,(HL)$ instruction. Again, except for YF and XF the flags are the same. YF and XF are copied from some sort of internal register. This register is related to 16 bit additions. Most instructions do not change this register. Unfortunately, I haven't tested all instructions yet, but here is the list so far.

ADD HL,xx Use the high byte of HL, ie. H before the addition.

LD $r,(IX+d)$ Use high byte of the resulting address IX+d.

JR d Use high byte target address of the jump.

LD r,r' Doesn't change this register.

Any help here would be most appreciated!

4.2 Memory Block Instructions [1]

The LDI/LDIR/LDD/LDDR instructions affect the flags in a strange way. At every iteration, a byte is copied. Take that byte and add the value of register A to it. Call that value n. Now, the flags are:

YF flag A copy of bit 1 of n.

HF flag Always reset.

XF flag A copy of bit 3 of n.

PF flag Set if BC not 0.

SF, ZF, CF flags These flags are unchanged.

And now for CPI/CPIR/CPD/CPDR. This instruction compares a series of bytes in memory to register A. Effectively, it can be said it does CP (HL) at every iteration. The result of that compare sets the HF flag, which is important for the next step. Take the value of register A, subtract the value of the memory address, and finally subtract the value of HF flag, which is set or reset by the hypothetical CP (HL). So, $n = A - (HL) - HF$.

SF, ZF, HF flags Set by the hypothetical CP (HL).

YF flag A copy of bit 1 of n.

XF flag A copy of bit 3 of n.

PF flag Set if BC is not 0.

NF flag Always set.

CF flag Unchanged.

4.3 I/O Block Instructions

These are the most be bizarre instructions, as far as flags is concerned. The PF flag is still unclear¹, and the HF and CF are most strange. The out instructions behave differently than the in instructions, which doesn't make the CPU very symmetrical.

First for the OUTI/OTIR/OUTD/OTDR instructions. At every iteration, a byte is read from memory and sent to an I/O port. Take that value (call it n), and add register L to it, *after* the HL register is increased (OUTI and OTIR) or decreased (OUTD and OTDR). If that calculation makes a carry (ie. it's larger than 255), both CF and HF are set. Otherwise CF and HF are reset.

¹Help would be most appreciated.

SF, ZF, YF, XF flags Affected by decreasing register B, as in DEC B.

CF, HF flags Set if $(L + n) > 255$.

NF flags A copy of bit 7 of n.

PF flag This flag is a mystery. It seems it is dependant on bits 2,1,0 of register C, bits 2,1,0 of n and register B.

The INI/INIR/IND/INDR instructions are slightly different. The value of CF and HF are computed differently. Call the value that is read from the I/O port n. Take the value of register C, and in the the case of INIR and INI, increase it by 1 and in the case of INDR and IND, decrease it by 1. Keep the result 8 bits — so it will be 0 if n was 255, with INI. Now add n to this; if it's larger than 255, set the CF and HF flags. Apart from that, the flags are the same.

4.4 16 bit I/O ports

Officially the Z80 has an 8 bit I/O port address space. When using the I/O ports, the 16 address lines are used. And in fact, the high 8 bit do actually have some value, so you can use 65536 ports after all. IN r, (C), OUT (C), r, and the Block I/O instructions actually place the entire BC register on the address bus. Similary IN A, (n) and OUT (n), A put $A \times 256 + n$ on the address bus.

4.5 Block Instructions

The repeated block instructions simply decrease the PC by two so the instruction is simply re-executed. So interrupts can occur during block instructions. So, LDIR is simply LDI + if BC is not 0, decrease PC by 2.

4.6 16 Bit Additions

The 16 bit additions are a bit more complicated. 16 bit additions are done in two stages: first the lower bytes are added, then the two higher bytes. The SF, YF, HF, XF flags are affected as by the second (high) 8 bit addition. ZF is set if the whole 16 bit result is 0.

4.7 DAA Instruction

This instruction is useful when you're using BCD values. After an addition or subtraction, DAA corrects the value back to BCD again. Note that it uses the CF flag, so it cannot be used after INC and DEC.

Here is the correction table, copied from the “Mostek Z80 Programming Manual”. This table is all I know about DAA operation.

| NF | CF | high nibble | HF | low nibble | # added to A | CF after execution |
|----|----|-------------|----|------------|--------------|--------------------|
| 0 | 0 | 0-9 | 0 | 0-9 | 00 | 0 |
| 0 | 0 | 0-8 | 0 | a-f | 06 | 0 |
| 0 | 0 | 0-9 | 1 | 0-3 | 06 | 0 |
| 0 | 0 | a-f | 0 | 0-9 | 60 | 1 |
| 0 | 0 | 9-f | 0 | a-f | 66 | 1 |
| 0 | 0 | a-f | 1 | 0-3 | 66 | 1 |
| 0 | 1 | 0-2 | 0 | 0-9 | 60 | 1 |
| 0 | 1 | 0-2 | 0 | a-f | 66 | 1 |
| 0 | 1 | 0-3 | 1 | 0-3 | 66 | 1 |
| 1 | 0 | 0-9 | 0 | 0-9 | 00 | 0 |
| 1 | 0 | 0-8 | 1 | 6-f | fa | 0 |
| 1 | 1 | 7-f | 0 | 0-9 | a0 | 1 |
| 1 | 1 | 6-f | 1 | 6-f | 9a | 1 |

Emulator builders: this instruction is hard to emulate. The Intel 80x86 equivalents are more or less the same, but differ for some input values. The best way to correctly emulate this instruction is by putting all possible output values (for A and F registers) in a table. The output depends on the following inputs: A register and HF, NF, CF flags. That's 11 bits in total, so the table will have size: $2^{11} \times 2 = 4096$ bytes, which isn't too bad.

Chapter 5

Interrupts

There are two types of interrupts, maskable and non-maskable. The maskable type is ignored if IFF1 is reset. Non-maskable interrupts (NMI) will be always accepted, and they have a higher priority, so if the two are requested at the same time the NMI will be accepted first.

For the interrupts, the following things are important: Interrupt Mode (set with the `IM 0`, `IM 1`, `IM 2` instructions), the interrupt flip-flops (IFF1 and IFF2), and the I register. When a maskable interrupt is accepted, an external device can put a value on the databus.

5.1 Non-Maskable Interrupts (NMI)

When a NMI is accepted, IFF1 is reset. At the end of the routine, IFF1 must be restored (so the running program is not affected). That's why IFF2 is there; to keep a copy of IFF1.

An NMI is accepted when the NMI pin on the Z80 is made low. The Z80 responds to the *change* of the line from +5 to 0 — so the interrupt line doesn't have a state, it's just a pulse. When this happens, a call is done to address 0066h and IFF1 is reset so the routine isn't bothered by maskable interrupts. The routine should end with an `RETN` (`RETurn from Nmi`) which is just a usual `RET`, but also copies IFF2 to IFF1, so the IFFs are the same as before the interrupt.

You can check whether interrupts were disabled or not during an NMI by using the `LD A,I` or `LD A,R` instruction. These instructions copy IFF2 to the PF flag.

5.2 Maskable Interrupts (INT)

If the INT line is low and IFF1 is set, a maskable interrupt is accepted — whether or not the last INT routine has finished. That's why you should not enable interrupts during such a routine, and make sure that the device that generated it has put the INT line up again before ending the routine. So unlike NMI interrupts, the interrupt line has a state; it's not a pulse.

When an INT is accepted, both IFF1 and IFF2 are cleared, preventing another interrupt from occurring which would end up as an infinite loop (and

overflowing the stack). What happens next depends on the Interrupt Mode.

A device can place a value on the databus when the interrupt is accepted. Some computer systems do not utilize this feature, and this value ends up being FFh.

Interrupt Mode 0 This is the 8080 compatibility mode. The instruction on the bus is executed (usually an `RST #` instruction, but it can be anything. The I register is not used.

Interrupt Mode 1 An `RST 38h` is executed, no matter what value is put on the bus or what value the I register has.

Interrupt Mode 2 A call is made to the address read from memory. What address is read from is calculated as follows: $(\text{I register}) \times 256 + (\text{value on bus})$. Of course a word (two bytes) are read, making the a address where the call is made to. In this way, you can have a vector table for interrupts.

At the end of a maskable interrupt, the interrupts should be enabled again. You can assume that was the state of the IFFs because otherwise the interrupt wasn't accepted. So, an INT routine always ends with an `EI` and a `RETI` (`RETI` according to the official documentation, more about that later):

```
INT:
    .
    .
    .
    EI
    RETI or RET
```

Note a fact about `EI`: a maskable interrupt isn't accepted directly after it, so the next opportunity for an interrupt is after the `RETI`. This is very useful; if the INT line is still low, an interrupt is accepted again. If this happens a lot and the interrupt is generated before the `RETI`, the stack could overflow (since the routine would be called again and again). But this property of `EI` prevents this.

`DI` is not necessary at the start of the interrupt routine: the interrupt flip-flops are cleared when accepting the interrupt.

You can use `RET` instead of `RETI`, depending on the hardware setup. `RETI` is only useful if you have something like a Z80 PIO to support daisy-chaining: queueing interrupts. The PIO can detect that the routine has ended by the opcode of `RETI`, and let another device generate an interrupt. That is why I called all the undocumented `EDxx RET` instructions `RETN`: All of them operate alike, the only difference of `RETI` is its specific opcode which the Z80 PIO recognises.

5.3 Things affecting the Interrupt flip-flops

All the IFF related things are:

| | IFF1 | IFF2 |
|-----------|------|------|
| CPU reset | 0 | 0 |

| | | | |
|---------------|------|---|----------------------------------|
| DI | 0 | 0 | |
| EI | 1 | 1 | |
| Accept INT | 0 | 0 | |
| Accept NMI | 0 | - | |
| RETI/N | IFF2 | - | All the EDxx RETI/N instructions |
| LD A,I/LD A,R | - | - | Copies IFF2 into PF flag |

If you're working with a Z80 system without NMIs (like the MSX), you can forget all about the two separate IFFs; since a NMI isn't ever generated, the two will always be the same.

Some documentation says that when an NMI is accepted, IFF1 is first copied into IFF2 before IFF1 is cleared. If this is true, the state of IFF2 is lost after a nested NMI, which is undesirable. Have tested this in the following way: make sure the Z80 is in EI mode, generate an NMI. In the NMI routine, wait for another NMI before executing RETN. In the second NMI IFF2 was still set, so IFF1 is *not* copied to IFF2 when accepting an NMI.

Another interesting fact is this. I was trying to figure out whether the undocumented ED RET instructions were RETN or RETI. I tested this by putting the machine in EI mode, wait for an NMI and end with one of the ED RET instructions. Then execute a HALT instruction. If IFF1 was not restored, the machine would hang but this did not happen with any of the instructions, including the documented RETI!

Since every INT routine must end with EI followed by RETI officially, It does not matter that RETI copies IFF2 into IFF1; both are set anyway.

5.4 HALT instruction

The HALT instruction halts the Z80; it does not increase the PC so that the instruction is re-executed, until a maskable or non-maskable interrupt is accepted. Only then does the Z80 increase the PC again and continues with the next instruction. During the HALT state, the HALT line is set. The PC is increased before the interrupt routine is called.

5.5 Where interrupts are accepted

During execution of instructions, interrupts won't be accepted. Only *between* instructions. This is also true for prefixed instructions.

Directly after an EI or DI instruction, interrupts aren't accepted. They're accepted again after the instruction after the EI (RET in the following example). So for example, look at this MSX2 routine that reads a scanline from the keyboard:

```
LD    C,A
DI
IN    A,(0AAh)
AND   0F0h
ADD   A,C
OUT   (0AAh),A
EI
```

```
IN    A, (0A9h)
RET
```

You can assume that there never is an interrupt after the EI, before the IN A, (0A9h) — which would be a problem because the MSX interrupt routine reads the keyboard too.

Using this feature of EI, it is possible to check whether it is true that interrupts are never accepted during instructions:

```
DI
  make sure INT is active
EI
  insert instruction to test
INT:
  store PC where INT was accepted
RET
```

And yes, for all instructions, including the prefixed ones, interrupts are never accepted during an instruction. Only after the tested instruction. Remember that block instructions simply re-execute themselves (by decreasing the PC with 2) so an interrupt is accepted after each iteration.

Another predictable test is this: at the “insert instruction to test” insert a large sequence of EI instructions. Of course, during execution of the EI instructions, no interrupts are accepted.

But now for the interesting stuff. ED or CB make up instructions, so interrupts are accepted after them. But DD and FD are prefixes, which only slightly affects the next opcode. If you test a large sequence of DDs or FDs, the same happens as with the EI instruction: no interrupts are accepted during the execution of these sequences.

This makes sense, if you think of DD and FD as a prefix which set the “use IX instead of HL” or “use IY instead of HL” flag. If an interrupt was accepted after DD or FD, this flag information would be lost, and:

```
DD 21 00 00    LD IX,0
```

could be interpreted as a simple LD HL,0 if the interrupt was after the last DD. Which never happens, so the implementation is correct. Although I haven’t tested this, as I imagine the same holds for NMI interrupts.

Chapter 6

Timing and R register

6.1 R register and memory refresh

During every first machine cycle (beginning of an instruction or part of it — prefixes have their own M1 two), the memory refresh cycle is issued. The whole IR register is put on the address bus, and the RFSH pin is lowered. It unclear whether the Z80 increases the R register before or after putting IR on the bus.

The R register is increased at every first machine cycle (M1). Bit 7 of the register is never changed by this; only the lower 7 bits are included in the addition. So bit 7 stays the same, but it can be changed using the LD R,A instruction.

Instructions without a prefix increase R by one. Instructions with an ED, CB, DD, FD prefix, increase R by two, and so do the DDCBxxxx and FD-CBxxxx instructions (weird enough). Just a stray DD or FD increases the R by one. LD A,R and LD R,A access the R register after it is increased (by the instruction itself).

Remember that the block instructions simply decrease the PC with two, so the instructions are re-executed. So LDIR increased R by BC times 2 (note that in the case of BC = 0, R is increased by 10000h times 2, effectively 0).

Accepting an maskable or non-maskable interrupt increases the R by one.

After a hardware reset, or after power on, the R register is reset to 0.

That should cover all there is to say about the R register. It is often used in programs for a random value, which is good but of course not truly random.

6.2 Undocumented Timings

1. Stray DD or FD: same as a NOP (increases R by one).
2. EDxx (not one of the list above or block instruction): Same as two NOPs. Increases R by 2.
3. Accepting interrupts (all one M1 and thus increase R by one): (T states)
 - IM 0: 13 (assuming the instruction is a RST)
 - IM 1: 13
 - IM 2: 19

NMI : 11

Chapter 7

Other Information

7.1 Errors in official documentation

In some official Zilog documentation, there are some errors. Some don't have all of these mistakes, so your documentation may not be flawed but these are just things to look out for.

- The Flag affection summary table shows that LDI/LDIR/LDD/LDDR instructions leave the SF and ZF in an undefined state. This is not correct; the SF and ZF flags are unaffected (like the same documentation says).
- Similarly, the same table shows that CPI/CPIR/CPD/CPDR leave the SF and HF flags in an undefined state. Not true, they are affected as defined elsewhere in the documentation.
- Also, the table says about INI/OUTD/etc “Z=0 if B <> 0 otherwise Z=0”; of course the latter should be Z=1.
- The INI/INIR/IND/INDR/OUTI/OUTD/OTIR/OTDR instructions do affect the CF flag (some official documentation says they leave it unaffected, important!) and the NF flag isn't always set but may also be reset (see above for exact operation).
- When an NMI is accepted, the IFF1 isn't copied to IFF2. Only IFF1 is reset.
- In the 8-bit Load Group, the last two bits of the second byte of the LD r, (IX + d) opcode should be 10 and not 01.
- In the 16-bit Arithmetic Group, bit 6 of the second byte of the ADD IX, pp opcode should be 0, not 1.
- IN x, (C) resets the HF flag, it never sets it. Some documentation states it is set according to the result of the operation; this is impossible since no arithmetic is done in this instruction.

Bibliography

- [1] Mark Rison Z80 page for !CPC.
<http://www.acorn.co.uk/mrison/en/cpc/tech.html>
- [2] YAZE (Yet Another Z80 Emulator). This is a CPM emulator by Frank Cringle. It emulates almost every undocumented flag, very good emulator. Also includes a very good instruction exerciser, and is released under the GPL.
<ftp://ftp.ping.de/pub/misc/emulators/yaze-1.10.tar.gz>
- [3] Z80 Family Official Support Page by Thomas Scherrer. Very good – your one-stop Z80 page.
http://www.geocities.com/SiliconValley/Peaks/3938/z80_home.htm
- [4] Spectrum FAQ technical information.
<http://www.void.jump.org/faq/tech.z80.html>
- [5] Gerton Lunter's Spectrum emulator (Z80). In the package there is a file TECHINFO.DOC, which contains a lot of interesting information. Note that the current version can only be unpacked in Windows.
<ftp://ftp.void.jump.org/pub/sinclair/emulators/pc/dos/z80-400.zip>
- [6] Mostek Z80 Programming Manual – a very good reference to the Z80.
- [7] Z80 Product Specification, from MSX2 Hardware Information.
<http://www.hardwareinfo.msx2.com/pdf/Zilog/z80.pdf>

Chapter 8

Instruction Tables

8.1 8-Bit Load Group

| Mnemonic | Symbolic Operation | Flags | | | | | | | | Opcode | | | Hex | Bytes | Cycles | M | T | Comments |
|-------------|--------------------|-------|----|----|----|----|----|----|----|--------|-----|-----|-----|-------|--------|----|-----------------------------|----------|
| | | SF | ZF | YF | HF | XF | PF | NF | CF | 76 | 543 | 210 | | | | | | |
| LD r,r' | r←r' | • | • | • | • | • | • | • | • | 01 | r | r' | | 1 | 1 | 4 | r,r' Reg | |
| LD p,p' | p←p' | • | • | • | • | • | • | • | • | 11 | 011 | 101 | DD | 2 | 2 | 8 | 000 B 001 C | |
| LD q,q' | q←q' | • | • | • | • | • | • | • | • | 11 | 111 | 101 | FD | 2 | 2 | 8 | 010 D 011 E | |
| LD r,n | r←n | • | • | • | • | • | • | • | • | 00 | r | 110 | | 2 | 2 | 7 | 100 H 101 L | |
| LD p,n | p←n | • | • | • | • | • | • | • | • | 11 | 011 | 101 | DD | 3 | 3 | 11 | 111 A | |
| LD q,n | q←n | • | • | • | • | • | • | • | • | 11 | 111 | 101 | FD | 3 | 3 | 11 | p,p' Reg 000 B 001 C | |
| LD r,(HL) | r←(HL) | • | • | • | • | • | • | • | • | 01 | r | 110 | | 1 | 2 | 7 | 010 D | |
| LD r,(IX+d) | r←(IX+d) | • | • | • | • | • | • | • | • | 11 | 011 | 101 | DD | 3 | 5 | 19 | 011 E 100 IXh 101 IXh | |
| LD r,(IY+d) | r←(IY+d) | • | • | • | • | • | • | • | • | 11 | 111 | 101 | FD | 3 | 5 | 19 | 111 A | |
| LD (HL),r | (HL)←r | • | • | • | • | • | • | • | • | 01 | 110 | r | | 1 | 2 | 7 | q,q' Reg | |
| LD (IX+d),r | (IX+d)←r | • | • | • | • | • | • | • | • | 11 | 011 | 101 | DD | 3 | 5 | 19 | 000 B 001 C 010 D | |
| LD (IY+d),r | (IY+d)←r | • | • | • | • | • | • | • | • | 11 | 111 | 101 | FD | 3 | 5 | 19 | 011 E 100 IYl 101 IYh | |
| LD (HL),n | (HL)←n | • | • | • | • | • | • | • | • | 00 | 110 | 110 | 36 | 2 | 3 | 10 | 111 A | |
| LD (IX+d),n | (IX+d)←n | • | • | • | • | • | • | • | • | 11 | 011 | 101 | DD | 4 | 5 | 19 | | |
| LD (IY+d),n | (IY+d)←n | • | • | • | • | • | • | • | • | 11 | 111 | 101 | FD | 4 | 5 | 19 | | |
| LD A,(BC) | A←(BC) | • | • | • | • | • | • | • | • | 00 | 001 | 010 | 0A | 1 | 2 | 7 | | |
| LD A,(DE) | A←(DE) | • | • | • | • | • | • | • | • | 00 | 011 | 010 | 1A | 1 | 2 | 7 | | |
| LD A,(nn) | A←(nn) | • | • | • | • | • | • | • | • | 00 | 111 | 010 | 3A | 3 | 4 | 13 | | |

(continued)

| Mnemonic | Symbolic Operation | Flags | | | | | | | | Opcode | | | Hex | Bytes | M Cycles | T States | Comments |
|-----------|---------------------|-------|----|----|----|----|------|----|----|----------------------------|-----|-----|-----|-------|----------|----------|----------|
| | | SF | ZF | YF | HF | XF | PF | NF | CF | 76 | 543 | 210 | | | | | |
| LD (BC),A | (BC) \leftarrow A | • | • | • | • | • | • | • | • | 00 | 000 | 010 | 02 | 1 | 2 | 7 | |
| LD (DE),A | (DE) \leftarrow A | • | • | • | • | • | • | • | • | 00 | 010 | 010 | 12 | 1 | 2 | 7 | |
| LD (nn),A | (nn) \leftarrow A | • | • | • | • | • | • | • | • | 00 | 110 | 010 | 32 | 3 | 4 | 13 | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| LD A,I | A \leftarrow I | ↓ | ↓ | ↓ | 0 | ↓ | IFF2 | 0 | • | 11 | 101 | 101 | ED | 2 | 2 | 9 | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| | | | | | | | | | | 01 | 010 | 111 | 57 | | | | |
| LD A,R | A \leftarrow R | ↓ | ↓ | ↓ | 0 | ↓ | IFF2 | 0 | • | 11 | 101 | 101 | ED | 2 | 2 | 9 | |
| | | | | | | | | | | 01 | 011 | 111 | 5F | | | | |
| LD I,A | I \leftarrow A | • | • | • | • | • | • | • | • | 11 | 101 | 101 | ED | 2 | 2 | 9 | |
| | | | | | | | | | | 01 | 000 | 111 | 47 | | | | |
| LD R,A | R \leftarrow A | • | • | • | • | • | • | • | • | 11 | 101 | 101 | ED | 2 | 2 | 9 | |
| | | | | | | | | | | 01 | 001 | 111 | 57 | | | | |

8.2 16-Bit Load Group

| Mnemonic | Symbolic Operation | Flags | | | | | | | | Opcode | | | Hex | Bytes | M Cycles | T States | Comments |
|------------|--|-------|----|----|----|----|----|----|----|----------------------------|-----|-----|-----|-------|----------|----------|----------|
| | | SF | ZF | YF | HF | XF | PF | NF | CF | 76 | 543 | 210 | | | | | |
| LD dd,nn | dd \leftarrow nn | • | • | • | • | • | • | • | • | 00 | dd0 | 001 | | 3 | 3 | 10 | dd Reg |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | 00 BC |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | 01 DE |
| LD IX,nn | IX \leftarrow nn | • | • | • | • | • | • | • | • | 11 | 011 | 101 | DD | 4 | 4 | 14 | 10 HL |
| | | | | | | | | | | 00 | 100 | 001 | 21 | | | | 11 SP |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| LD IY,nn | IY \leftarrow nn | • | • | • | • | • | • | • | • | 11 | 111 | 101 | FD | 4 | 4 | 14 | |
| | | | | | | | | | | 00 | 100 | 001 | 21 | | | | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| LD HL,(nn) | H \leftarrow (nn+1) L \leftarrow (nn) | • | • | • | • | • | • | • | • | 00 | 101 | 010 | 2A | 3 | 5 | 16 | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| LD dd,(nn) | ddh \leftarrow (nn+1) ddl \leftarrow (nn) | • | • | • | • | • | • | • | • | 11 | 101 | 101 | ED | 4 | 6 | 20 | |
| | | | | | | | | | | 01 | dd1 | 011 | | | | | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| LD IX,(nn) | IXh \leftarrow (nn+1) IXl \leftarrow (nn) | • | • | • | • | • | • | • | • | 11 | 011 | 101 | DD | 4 | 6 | 20 | |
| | | | | | | | | | | 00 | 101 | 010 | 2A | | | | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| LD IY,(nn) | IYh \leftarrow (nn+1) IYl \leftarrow (nn) | • | • | • | • | • | • | • | • | 11 | 111 | 101 | FD | 4 | 6 | 20 | |
| | | | | | | | | | | 00 | 101 | 010 | 2A | | | | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| LD (nn),HL | (nn+1) \leftarrow H (nn) \leftarrow L | • | • | • | • | • | • | • | • | 00 | 100 | 010 | 22 | 3 | 5 | 16 | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| LD (nn),dd | (nn+1) \leftarrow ddh (nn) \leftarrow ddl | • | • | • | • | • | • | • | • | 11 | 101 | 101 | ED | 4 | 6 | 20 | |
| | | | | | | | | | | 01 | dd0 | 011 | | | | | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| LD (nn),IX | (nn+1) \leftarrow IXh (nn) \leftarrow IXl | • | • | • | • | • | • | • | • | 11 | 011 | 101 | DD | 4 | 6 | 20 | |
| | | | | | | | | | | 00 | 100 | 010 | 22 | | | | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| LD (nn),IY | (nn+1) \leftarrow IYh (nn) \leftarrow IYl | • | • | • | • | • | • | • | • | 11 | 111 | 101 | FD | 4 | 6 | 20 | |
| | | | | | | | | | | 00 | 100 | 010 | 22 | | | | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| | | | | | | | | | | $\leftarrow n \rightarrow$ | | | | | | | |
| LD SP,HL | SP \leftarrow HL | • | • | • | • | • | • | • | • | 11 | 111 | 001 | F9 | 1 | 1 | 6 | |
| LD SP,IX | SP \leftarrow IX | • | • | • | • | • | • | • | • | 11 | 011 | 101 | DD | 2 | 2 | 10 | |
| | | | | | | | | | | 11 | 111 | 001 | F9 | | | | |
| LD SP,IY | SP \leftarrow IY | • | • | • | • | • | • | • | • | 11 | 111 | 101 | FD | 2 | 2 | 10 | |
| | | | | | | | | | | 11 | 111 | 001 | F9 | | | | |

(continued)

8.3 Exchange, Block Transfer, Block Search Group

| Mnemonic | Symbolic Operation | Flags | | | | | | | | Opcode | | | Hex | Bytes | M Cycles | T States | Comments |
|------------|--|--------------|--------------|--------------|--------------|--------------|----------------|----|----|--------|-----|-----|-----|-------|----------|----------|-------------------------------------|
| | | SF | ZF | YF | HF | XF | PF | NF | CF | 76 | 543 | 210 | | | | | |
| EX DE,HL | DE \leftrightarrow HL | • | • | • | • | • | • | • | • | 11 | 101 | 011 | E8 | 1 | 1 | 4 | |
| EX AF,AF' | AF \leftrightarrow AF' | • | • | • | • | • | • | • | • | 00 | 001 | 000 | 08 | 1 | 1 | 4 | |
| EXX | BC \leftrightarrow BC' DE \leftrightarrow DE' HL \leftrightarrow HL' | • | • | • | • | • | • | • | • | 11 | 011 | 001 | D8 | 1 | 1 | 4 | |
| EX (SP),HL | H \leftrightarrow (SP+1) L \leftrightarrow (SP) | • | • | • | • | • | • | • | • | 11 | 100 | 011 | E3 | 1 | 5 | 19 | |
| EX (SP),IX | IXh \leftrightarrow (SP+1) IXl \leftrightarrow (SP) | • | • | • | • | • | • | • | • | 11 | 011 | 101 | DD | 2 | 6 | 23 | |
| EX (SP),IY | IYh \leftrightarrow (SP+1) IYl \leftrightarrow (SP) | • | • | • | • | • | • | • | • | 11 | 111 | 101 | FD | 2 | 6 | 23 | |
| LDI | (DE) \leftarrow (HL) DE \leftarrow DE+1 HL \leftarrow HL+1 BC \leftarrow BC-1 | • | • | \uparrow^4 | 0 | \uparrow^4 | \uparrow^1 | 0 | • | 11 | 101 | 101 | ED | 2 | 4 | 16 | |
| LDIR | (DE) \leftarrow (HL) DE \leftarrow DE+1 HL \leftarrow HL+1 BC \leftarrow BC-1 Repeat until BC=0 | • | • | \uparrow^4 | 0 | \uparrow^4 | 0 ² | 0 | • | 11 | 101 | 101 | ED | 2 | 5 | 21 | if BC \neq 0 |
| | | | | | | | | | | 10 | 110 | 000 | B0 | 2 | 4 | 16 | if BC=0 |
| LDD | (DE) \leftarrow (HL) DE \leftarrow DE-1 HL \leftarrow HL-1 BC \leftarrow BC-1 | • | • | \uparrow^4 | 0 | \uparrow^4 | \uparrow^1 | 0 | • | 11 | 101 | 101 | ED | 2 | 4 | 16 | |
| | | | | | | | | | | 10 | 101 | 000 | A8 | | | | |
| LDDR | (DE) \leftarrow (HL) DE \leftarrow DE-1 HL \leftarrow HL-1 BC \leftarrow BC-1 Repeat until BC=0 | • | • | \uparrow^4 | 0 | \uparrow^4 | 0 ² | 0 | • | 11 | 101 | 101 | ED | 2 | 5 | 21 | if BC \neq 0 |
| | | | | | | | | | | 10 | 111 | 000 | B8 | 2 | 4 | 16 | if BC=0 |
| CPI | A-(HL) HL \leftarrow HL+1 BC \leftarrow BC-1 | \uparrow^4 | \uparrow^3 | \uparrow^4 | \uparrow^4 | \uparrow^4 | \uparrow^1 | 1 | • | 11 | 101 | 101 | ED | 2 | 4 | 16 | |
| | | | | | | | | | | 10 | 100 | 001 | A1 | | | | |
| CPIR | A-(HL) HL \leftarrow HL+1 BC \leftarrow BC-1 Repeat until A=(HL) or BC=0 | \uparrow^4 | \uparrow^3 | \uparrow^4 | \uparrow^4 | \uparrow^4 | \uparrow^1 | 1 | • | 11 | 101 | 101 | ED | 2 | 5 | 21 | if BC \neq 0 and A \neq (HL) |
| | | | | | | | | | | 10 | 110 | 001 | B1 | 2 | 4 | 18 | if BC=0 or A=(HL) |
| CPD | A-(HL) HL \leftarrow HL-1 BC \leftarrow BC-1 | \uparrow^4 | \uparrow^3 | \uparrow^4 | \uparrow^4 | \uparrow^4 | \uparrow^1 | 1 | • | 11 | 101 | 101 | ED | 2 | 4 | 16 | |
| | | | | | | | | | | 10 | 101 | 001 | A9 | | | | |
| CPDR | A-(HL) HL \leftarrow HL-1 BC \leftarrow BC-1 Repeat until A=(HL) or BC=0 | \uparrow^4 | \uparrow^3 | \uparrow^4 | \uparrow^4 | \uparrow^4 | \uparrow^1 | 1 | • | 11 | 101 | 101 | ED | 2 | 5 | 21 | if BC \neq 0 and A \neq (HL) |
| | | | | | | | | | | 10 | 111 | 001 | B9 | 2 | 4 | 18 | if BC=0 or A=(HL) |

Note: ¹PF is 0 the result of BC-1=0, otherwise PF is set.
²PF is 0 only at completion of the instruction.
³ZF is set if A=(HL), otherwise ZF is reset.
⁴See section 4.2 for a description.

8.4 8-Bit Arithmetic and Logical Group

| Mnemonic | Symbolic Operation | Flags | | | | | | | | Opcode | | | Hex | Bytes | M Cycles | T States | Comments |
|--------------|--------------------|-------|----|----|----|----|----|----|----|--------|------------|------------|-----|-------|----------|----------|----------|
| | | SF | ZF | YF | HF | XF | PF | NF | CF | 76 | 543 | 210 | | | | | |
| ADD A,r | A←A+r | ↓ | ↓ | ↓ | ↓ | ↓ | VF | 0 | ↓ | 10 | <u>000</u> | r | DD | 1 | 1 | 4 | r Reg |
| ADD A,p | A←A+p | ↓ | ↓ | ↓ | ↓ | ↓ | VF | 0 | ↓ | 11 | 011 | 101 | DD | 2 | 2 | 8 | 000 B |
| | | | | | | | | | | 10 | <u>000</u> | p | | | | | 001 C |
| ADD A,q | A←A+q | ↓ | ↓ | ↓ | ↓ | ↓ | VF | 0 | ↓ | 11 | 111 | 101 | FD | 2 | 2 | 8 | 010 D |
| | | | | | | | | | | 10 | <u>000</u> | q | | | | | 011 E |
| ADD A,n | A←A+n | ↓ | ↓ | ↓ | ↓ | ↓ | VF | 0 | ↓ | 11 | <u>000</u> | ← n → | | 2 | 2 | 7 | 100 H |
| | | | | | | | | | | 10 | <u>000</u> | 110 | | | | | 101 L |
| ADD A,(HL) | A←A+(HL) | ↓ | ↓ | ↓ | ↓ | ↓ | VF | 0 | ↓ | 10 | <u>000</u> | 110 | | 1 | 2 | 7 | 111 A |
| ADD A,(IX+d) | A←A+(IX+d) | ↓ | ↓ | ↓ | ↓ | ↓ | VF | 0 | ↓ | 11 | 011 | 101 | DD | 3 | 5 | 19 | |
| | | | | | | | | | | 10 | <u>000</u> | 110 | | | | | |
| | | | | | | | | | | ← d → | | | | | | | p Reg |
| ADD A,(IY+d) | A←A+(IY+d) | ↓ | ↓ | ↓ | ↓ | ↓ | VF | 0 | ↓ | 11 | 111 | 101 | FD | 3 | 5 | 19 | 000 B |
| | | | | | | | | | | 10 | <u>000</u> | 110 | | | | | 001 C |
| | | | | | | | | | | ← d → | | | | | | | 010 D |
| ADC A,s | A←A+s+CF | ↓ | ↓ | ↓ | ↓ | ↓ | VF | 0 | ↓ | | <u>001</u> | | | | | | 011 E |
| SUB s | A←A-s | ↓ | ↓ | ↓ | ↓ | ↓ | VF | 1 | ↓ | | <u>010</u> | | | | | | 100 IXh |
| SBC s | A←A-s-CF | ↓ | ↓ | ↓ | ↓ | ↓ | VF | 1 | ↓ | | <u>011</u> | | | | | | 101 IXl |
| AND s | A←A∧s | ↓ | ↓ | ↓ | ↓ | 1 | ↓ | PF | 0 | 0 | <u>100</u> | | | | | | 111 A |
| OR s | A←A∨s | ↓ | ↓ | ↓ | ↓ | 0 | ↓ | PF | 0 | 0 | <u>110</u> | | | | | | |
| XOR s | A←A⊙s | ↓ | ↓ | ↓ | ↓ | 0 | ↓ | PF | 0 | 0 | <u>101</u> | | | | | | |
| CP s | A-s | ↓ | ↓ | ↓ | ↓ | ↓ | VF | 1 | ↓ | | <u>111</u> | | | | | | q Reg |
| INC r | r←r+1 | ↓ | ↓ | ↓ | ↓ | ↓ | VF | 0 | • | 00 | r | <u>100</u> | | 1 | 1 | 4 | 000 B |
| INC p | p←p+1 | ↓ | ↓ | ↓ | ↓ | ↓ | VF | 0 | • | 11 | 011 | 101 | DD | 2 | 2 | 8 | 001 C |
| | | | | | | | | | | 00 | p | <u>100</u> | | | | | 010 D |
| INC q | q←q+1 | ↓ | ↓ | ↓ | ↓ | ↓ | VF | 0 | • | 11 | 111 | 101 | FD | 2 | 2 | 8 | 011 E |
| | | | | | | | | | | 00 | q | <u>100</u> | | | | | 100 IYh |
| INC (HL) | (HL)←(HL)+1 | ↓ | ↓ | ↓ | ↓ | ↓ | VF | 0 | • | 00 | 110 | <u>100</u> | | 1 | 3 | 11 | 101 IYl |
| INC (IX+d) | (IX+d)←(IX+d)+1 | ↓ | ↓ | ↓ | ↓ | ↓ | VF | 0 | • | 11 | 011 | 101 | DD | 3 | 6 | 26 | 111 A |
| | | | | | | | | | | 00 | 110 | <u>100</u> | | | | | |
| | | | | | | | | | | ← d → | | | | | | | |
| INC (IY+d) | (IY+d)←(IY+d)+1 | ↓ | ↓ | ↓ | ↓ | ↓ | VF | 0 | • | 11 | 111 | 101 | FD | 3 | 6 | 26 | |
| | | | | | | | | | | 00 | 110 | <u>100</u> | | | | | |
| | | | | | | | | | | ← d → | | | | | | | |
| DEC m | m←m-1 | ↓ | ↓ | ↓ | ↓ | ↓ | VF | 1 | • | | | <u>101</u> | | | | | |

Note: ¹YF and XF flags are copied from the operand s, not the result A-s
s is any of r, p, q, n, (HL), (IX+d), (IY+d) as shown for ADD. The indicated bits replace the 000 in the ADD set above
m is any of r, p, q, (HL), (IX+d), (IY+d) as shown for INC. Replace 100 with 101 in opcode

8.5 General-Purpose Arithmetic and CPU Control Group

| Mnemonic | Symbolic Operation | Flags | | | | | | | | Opcode | | | Hex | Bytes | M Cycles | T States | Comments |
|-------------------|--------------------|-------|----|----------------|----------------|----------------|----|----|----|--------|-----|-----|-----|-------|----------|----------|----------------------------|
| | | SF | ZF | YF | HF | XF | PF | NF | CF | 76 | 543 | 210 | | | | | |
| DAA | | ↓ | ↓ | ↓ | ↓ | ↓ | PF | • | ↓ | 00 | 100 | 111 | 27 | 1 | 1 | 4 | Decimal adjust accumulator |
| CPL | A← \bar{A} | • | • | ↓ | 1 | ↓ | • | 1 | • | 00 | 101 | 111 | 2F | 1 | 1 | 4 | Compliment |
| NEG | A←0-A | ↓ | ↓ | ↓ | ↓ | ↓ | VF | 1 | ↓ | 11 | 011 | 101 | ED | 2 | 2 | 8 | Negate |
| | | | | | | | | | | 01 | 000 | 100 | 44 | | | | |
| CCF | CF← \bar{CF} | • | • | ↓ ¹ | ↓ ² | ↓ ¹ | • | 0 | ↓ | 00 | 111 | 111 | 3F | 1 | 1 | 4 | |
| SCF | CF←1 | • | • | ↓ ¹ | 0 | ↓ ¹ | • | 0 | 1 | 00 | 110 | 111 | 37 | 1 | 1 | 4 | |
| NOP | | • | • | • | • | • | • | • | • | 00 | 000 | 000 | 00 | 1 | 1 | 4 | |
| HALT | | • | • | • | • | • | • | • | • | 01 | 110 | 110 | 76 | 1 | 1 | 4 | |
| DI ³ | IFF1,2←0 | • | • | • | • | • | • | • | • | 11 | 110 | 011 | F3 | 1 | 1 | 4 | |
| EI ³ | IFF1,2←1 | • | • | • | • | • | • | • | • | 11 | 111 | 011 | FB | 1 | 1 | 4 | |
| IM 0 ⁴ | | • | • | • | • | • | • | • | • | 11 | 011 | 101 | ED | 2 | 2 | 8 | |
| | | | | | | | | | | 01 | 000 | 110 | 46 | | | | |
| IM 1 ⁴ | | • | • | • | • | • | • | • | • | 11 | 011 | 101 | ED | 2 | 2 | 8 | |
| | | | | | | | | | | 01 | 010 | 110 | 56 | | | | |
| IM 2 ⁴ | | • | • | • | • | • | • | • | • | 11 | 011 | 101 | ED | 2 | 2 | 8 | |
| | | | | | | | | | | 01 | 011 | 110 | 5E | | | | |

Note: ¹YF and XF are copied from register A.
²HF is like CF before the instruction.
³No interrupts are accepted directly after EI or DI.
⁴This instruction has other undocumented opcodes.

8.6 16-Bit Arithmetic Group

| Mnemonic | Symbolic Operation | Flags | | | | | | | Opcode | | | Hex | Bytes | M Cycles | T States | Comments | |
|-----------|--------------------|----------------|----------------|----------------|----------------|----------------|-----------------|----|----------------|----|-----|-----|-------|----------|----------|----------|----------------|
| | | SF | ZF | YF | HF | XF | PF | NF | CF | 76 | 543 | | | | | | 210 |
| ADD HL,ss | HL←HL+ss | • | • | ↑ ² | ↑ ² | ↑ ² | • | 0 | ↑ ¹ | 01 | ss1 | 001 | | 1 | 3 | 11 | ss Reg |
| ADC HL,ss | HL←HL+ss+CF | ↑ ¹ | ↑ ¹ | ↑ ² | ↑ ² | ↑ ² | VF ¹ | 0 | ↑ ¹ | 11 | 011 | 101 | ED | 2 | 4 | 15 | 00 BC 01 DE |
| SBC HL,ss | HL←HL-ss-CF | ↑ ¹ | ↑ ¹ | ↑ ² | ↑ ² | ↑ ² | VF ¹ | 0 | ↑ ¹ | 11 | 011 | 101 | ED | 2 | 4 | 15 | 10 HL 11 SP |
| ADD IX,pp | IX←IX+pp | • | • | ↑ ² | ↑ ² | ↑ ² | • | 0 | ↑ ¹ | 11 | 011 | 110 | DD | 2 | 4 | 15 | pp Reg |
| ADD IY,qq | IY←IY+qq | • | • | ↑ ² | ↑ ² | ↑ ² | • | 0 | ↑ ¹ | 11 | 111 | 110 | FD | 2 | 4 | 15 | 00 BC 01 DE |
| INC ss | ss←ss+1 | • | • | • | • | • | • | • | • | 00 | ss0 | 011 | | 1 | 1 | 6 | 10 IX |
| INC IX | IX←IX+1 | • | • | • | • | • | • | • | • | 11 | 011 | 101 | DD | 2 | 2 | 10 | 11 SP |
| INC IY | IY←IY+1 | • | • | • | • | • | • | • | • | 11 | 111 | 101 | FD | 2 | 2 | 10 | qq Reg |
| DEC ss | ss←ss-1 | • | • | • | • | • | • | • | • | 00 | ss1 | 011 | | 1 | 1 | 6 | 01 DE |
| DEC IX | IX←IX-1 | • | • | • | • | • | • | • | • | 11 | 011 | 101 | DD | 2 | 2 | 10 | 10 HL 11 SP |
| DEC IY | IY←IY-1 | • | • | • | • | • | • | • | • | 11 | 111 | 101 | FD | 2 | 2 | 10 | |
| | | | | | | | | | | 00 | 101 | 011 | 2B | | | | |

Note: ¹Flag is affected by the 16 bit result.
²Flag is affected by the high-byte addition.

CHAPTER 8. INSTRUCTION TABLES

8.7 Rotate and Shift Group

| Mnemonic | Symbolic Operation | Flags | | | | | | | | Opcode | | | | M Bytes | T Cycles | States | Comments |
|---------------|-----------------------------------|-------|----|----|----|----|----|----|----|----------------|-------------------|-------------------|----------|---------|----------|--------|----------------------------------|
| | | SF | ZF | YF | HF | XF | PF | NF | CF | 76 | 543 | 210 | Hex | | | | |
| RLCA | | • | • | ↓ | 0 | ↓ | • | 0 | ↓ | 00 | 000 | 111 | 07 | 1 | 1 | 4 | |
| RLA | | • | • | ↓ | 0 | ↓ | • | 0 | ↓ | 00 | 010 | 111 | 17 | 1 | 1 | 4 | |
| RRCA | | • | • | ↓ | 0 | ↓ | • | 0 | ↓ | 00 | 001 | 111 | 0F | 1 | 1 | 4 | |
| RRA | | • | • | ↓ | 0 | ↓ | • | 0 | ↓ | 00 | 011 | 111 | 1F | 1 | 1 | 4 | |
| RLC r | | ↓ | ↓ | ↓ | 0 | ↓ | PF | 0 | ↓ | 11 00 | 001 000 | 011 r | CB | 2 | 2 | 8 | r Reg 000 B |
| RLC (HL) | | ↓ | ↓ | ↓ | 0 | ↓ | PF | 0 | ↓ | 11 00 | 001 000 | 011 110 | CB | 2 | 4 | 15 | 001 C 010 D |
| RLC (IX+d) | | ↓ | ↓ | ↓ | 0 | ↓ | PF | 0 | ↓ | 11 11 00 | 011 001 000 | 101 011 110 | DD CB | 4 | 6 | 23 | 011 E 100 H 101 L 111 A |
| RLC (IY+d) | | ↓ | ↓ | ↓ | 0 | ↓ | PF | 0 | ↓ | 11 11 00 | 111 001 000 | 101 011 110 | FD CB | 4 | 6 | 23 | |
| RLC (IX+d), r | r ← (IX+d) RLC r (IX+d) ← r | ↓ | ↓ | ↓ | 0 | ↓ | PF | 0 | ↓ | 11 11 00 | 011 001 000 | 101 011 r | DD CB | 4 | 6 | 23 | |
| RLC (IY+d), r | r ← (IY+d) RLC r (IY+d) ← r | ↓ | ↓ | ↓ | 0 | ↓ | PF | 0 | ↓ | 11 11 00 | 111 001 000 | 101 011 r | FD CB | 4 | 6 | 23 | |
| RL m | | ↓ | ↓ | ↓ | 0 | ↓ | PF | 0 | ↓ | | | | 010 | | | | |
| RRC m | | ↓ | ↓ | ↓ | 0 | ↓ | PF | 0 | ↓ | | | | 001 | | | | |
| RR m | | ↓ | ↓ | ↓ | 0 | ↓ | PF | 0 | ↓ | | | | 011 | | | | |
| SLA m | | ↓ | ↓ | ↓ | 0 | ↓ | PF | 0 | ↓ | | | | 100 | | | | |
| SLL m | | ↓ | ↓ | ↓ | 0 | ↓ | PF | 0 | ↓ | | | | 110 | | | | |
| SRA m | | ↓ | ↓ | ↓ | 0 | ↓ | PF | 0 | ↓ | | | | 101 | | | | |
| SLA m | | ↓ | ↓ | ↓ | 0 | ↓ | PF | 0 | ↓ | | | | 111 | | | | |
| RLD | | ↓ | ↓ | ↓ | 0 | ↓ | PF | 0 | • | 11 01 | 101 101 | 101 111 | ED 6F | 2 | 5 | 18 | |
| RRD | | ↓ | ↓ | ↓ | 0 | ↓ | PF | 0 | • | 11 01 | 101 101 | 101 111 | ED 67 | 2 | 5 | 18 | |

Note: m is one of r, (HL), (IX+d), (IY+d). To form new opcode replace 000 of RLCs with shown code.

CHAPTER 8. INSTRUCTION TABLES

8.8 Bit Set, Reset and Test Group

| Mnemonic | Symbolic Operation | Flags | | | | | | | | Opcode | | | | M | T | Comments | |
|---------------------------|--|----------------|----|----------------|----|----------------|----------------|----|----|--------|-----|-----|-----|---|---|----------|----------------------------------|
| | | SF | ZF | YF | HF | XF | PF | NF | CF | 76 | 543 | 210 | Hex | | | | Bytes |
| BIT b,r | ZF ← $\overline{r_b}$ | ↑ ¹ | ↓ | ↑ ¹ | 1 | ↑ ¹ | ↑ ¹ | 0 | • | 11 | 001 | 011 | CB | 2 | 2 | 8 | r Reg 000 B |
| BIT b,(HL) | ZF ← $\overline{(HL)_b}$ | ↑ ¹ | ↓ | ↑ ¹ | 1 | ↑ ¹ | ↑ ¹ | 0 | • | 11 | 001 | 011 | CB | 2 | 3 | 12 | 001 C 010 D |
| BIT b,(IX+d) ² | ZF ← $\overline{(IX+d)_b}$ | ↑ ¹ | ↓ | ↑ ¹ | 1 | ↑ ¹ | ↑ ¹ | 0 | • | 11 | 011 | 101 | DD | 4 | 8 | 20 | 011 E 100 H 101 L 111 A |
| BIT b,(IY+d) ² | ZF ← $\overline{(IY+d)_b}$ | ↑ ¹ | ↓ | ↑ ¹ | 1 | ↑ ¹ | ↑ ¹ | 0 | • | 11 | 111 | 101 | FD | 4 | 8 | 20 | |
| SET b,r | $r_b \leftarrow 1$ | • | • | • | • | • | • | • | • | 11 | 001 | 011 | CB | 2 | 2 | 8 | b Bit 000 0 |
| SET b,(HL) | $(HL)_b \leftarrow 1$ | • | • | • | • | • | • | • | • | 11 | 001 | 011 | CB | 2 | 4 | 15 | 001 1 010 2 |
| SET b,(IX+d) | $(IX+d)_b \leftarrow 1$ | • | • | • | • | • | • | • | • | 11 | 011 | 101 | DD | 4 | 8 | 23 | 011 3 100 4 101 5 110 6 |
| SET b,(IY+d) | $(IY+d)_b \leftarrow 1$ | • | • | • | • | • | • | • | • | 11 | 111 | 101 | FD | 4 | 8 | 23 | 111 7 |
| SET b,(IX+d),r | $r \leftarrow (IX+d)$ $r_b \leftarrow 1$ $(IX+d) \leftarrow r$ | • | • | • | • | • | • | • | • | 11 | 011 | 101 | DD | 4 | 8 | 23 | |
| SET b,(IY+d),r | $r \leftarrow (IY+d)$ $r_b \leftarrow 1$ $(IY+d) \leftarrow r$ | • | • | • | • | • | • | • | • | 11 | 111 | 101 | FD | 4 | 8 | 23 | |
| RES b,m | $m_b \leftarrow 0$ | • | • | • | • | • | • | • | • | 11 | 10 | | | | | | |

Note: ¹See section 4.1 for a complete description.
²Instruction has other undocumented opcodes.
m is one of r, (HL), (IX+d), (IY+d). To form RES instruction, replace **11** with **10**.

8.9 Jump Group

| Mnemonic | Symbolic Operation | Flags | | | | | | | | Opcode | | | | M | T | Comments | | |
|----------|----------------------------------|-------|----|----|----|----|----|----|----|--------|-----|-----|-----|----|---|----------|---------------------------------|--------------------------|
| | | SF | ZF | YF | HF | XF | PF | NF | CF | 76 | 543 | 210 | Hex | | | | Bytes | Cycles |
| JP nn | PC ← nn | • | • | • | • | • | • | • | • | 11 | 000 | 011 | C3 | 3 | 3 | 10 | cc Condition 000 NZ 001 Z | |
| JP cc,nn | if cc PC ← nn | • | • | • | • | • | • | • | • | 11 | cc | 010 | | 3 | 3 | 10 | 010 NC 011 C 100 PO | |
| JR e | PC ← PC+e | • | • | • | • | • | • | • | • | 00 | 011 | 000 | | 18 | 2 | 3 | 12 | 101 PE 110 P 111 M |
| JR ss,e | if ss PC ← PC+e | • | • | • | • | • | • | • | • | 00 | 1ss | 000 | | 2 | 3 | 12 | if ss is true | |
| JP (HL) | PC ← HL | • | • | • | • | • | • | • | • | 11 | 101 | 001 | E9 | 1 | 1 | 4 | if ss is false | |
| JP (IX) | PC ← IX | • | • | • | • | • | • | • | • | 11 | 011 | 101 | DD | 2 | 2 | 8 | ss Condition 11 C | |
| JP (IY) | PC ← IY | • | • | • | • | • | • | • | • | 11 | 111 | 101 | FD | 2 | 2 | 8 | 10 NC 01 Z 00 NZ | |
| DJNZ e | B ← B-1 if B ≠ 0 PC ← PC+e | • | • | • | • | • | • | • | • | 00 | 010 | 000 | | 10 | 2 | 2 | 10 | if B=0 |
| | | | | | | | | | | | | | | 2 | 3 | 13 | if B ≠ 0 | |

Note: e is a signed two-compliment in the range -127, 129.
e-2 in the opcode provides an effective number of PC+e as PC is incremented by two prior to the addition of e.

CHAPTER 8. INSTRUCTION TABLES

8.10 Call and Return Group

| Mnemonic | Symbolic Operation | Flags | | | | | | | | Opcode | | | Hex | Bytes | M Cycles | T States | Comments |
|-------------------|---|-------|----|----|----|----|----|----|----|--------|-----|-----|-----|-------|----------|----------|----------------|
| | | SF | ZF | YF | HF | XF | PF | NF | CF | 76 | 543 | 210 | | | | | |
| CALL nn | (SP-1)←PCh (SP-2)←PCl SP←SP-2 PC←nn | • | • | • | • | • | • | • | • | 11 | 001 | 101 | CD | 3 | 5 | 17 | |
| CALL cc,nn | if cc is true (SP-1)←PCh (SP-2)←PCl SP←SP-2 PC←nn | • | • | • | • | • | • | • | • | 11 | cc | 100 | | 3 | 3 | 10 | if cc is false |
| | (SP-1)←PCh (SP-2)←PCl SP←SP-2 PC←nn | | | | | | | | | | | | | 3 | 5 | 17 | if cc is true |
| RET | PCl←(SP) PCh←(SP+1) SP←SP+2 | • | • | • | • | • | • | • | • | 11 | 001 | 001 | C9 | 1 | 3 | 10 | |
| RET cc | if cc is true PCl←(SP) PCh←(SP+1) SP←SP+2 | • | • | • | • | • | • | • | • | 11 | cc | 000 | | 1 | 1 | 5 | if cc is false |
| | PCl←(SP) PCh←(SP+1) SP←SP+2 | | | | | | | | | | | | | 1 | 5 | 17 | if cc is true |
| RETI ¹ | PCl←(SP) PCh←(SP+1) SP←SP+2 | • | • | • | • | • | • | • | • | 11 | 101 | 101 | ED | 2 | 4 | 14 | cc Condition |
| | PCl←(SP) PCh←(SP+1) SP←SP+2 | | | | | | | | | 01 | 001 | 101 | 4D | | | | 000 NZ |
| | PCl←(SP) PCh←(SP+1) SP←SP+2 | | | | | | | | | | | | | | | | 001 Z |
| RETN ² | PCl←(SP) PCh←(SP+1) SP←SP+2 IFF1←IFF2 | • | • | • | • | • | • | • | • | 11 | 101 | 101 | ED | 2 | 4 | 14 | 010 NC |
| | PCl←(SP) PCh←(SP+1) SP←SP+2 IFF1←IFF2 | | | | | | | | | 01 | 000 | 101 | 45 | | | | 011 C |
| | | | | | | | | | | | | | | | | | 100 PO |
| | | | | | | | | | | | | | | | | | 101 PE |
| | | | | | | | | | | | | | | | | | 110 P |
| | | | | | | | | | | | | | | | | | 111 M |
| RST p | (SP-1)←PCh (SP-2)←PCl SP←SP-2 PC←p | • | • | • | • | • | • | • | • | 11 | t | 111 | | 1 | 3 | 11 | t p |
| | (SP-1)←PCh (SP-2)←PCl SP←SP-2 PC←p | | | | | | | | | | | | | | | | 000 0h |
| | | | | | | | | | | | | | | | | | 001 8h |
| | | | | | | | | | | | | | | | | | 010 10h |
| | | | | | | | | | | | | | | | | | 011 18h |
| | | | | | | | | | | | | | | | | | 100 20h |
| | | | | | | | | | | | | | | | | | 101 28h |
| | | | | | | | | | | | | | | | | | 110 30h |
| | | | | | | | | | | | | | | | | | 111 38h |

¹ RETI also copies IFF2 into IFF1, like RETN.

² This instruction has other undocumented opcodes.

8.11 Input and Output Group

| Mnemonic | Symbolic Operation | Flags | | | | | | | | Opcode | | | Hex | Bytes | M Cycles | T States | Comments | |
|-----------|---|----------------|----------------|----------------|----------------|----------------|----|----------------|----------------|--------|-----|-----|-----|-------|----------|----------|----------|-----|
| | | SF | ZF | YF | HF | XF | PF | NF | CF | 76 | 543 | 210 | | | | | r | Reg |
| IN A,(n) | A←(n) | • | • | • | • | • | • | • | • | 11 | 011 | 011 | DB | 2 | 3 | 11 | 000 | B |
| | | | | | | | | | | ← n → | | | | | | | | |
| IN r,(C) | r←(C) | ↓ | ↓ | ↓ | 0 | ↓ | PF | 0 | • | 11 | 101 | 101 | ED | 2 | 3 | 12 | 001 | C |
| | | | | | | | | | | 01 | r | 000 | | | | | 010 | D |
| IN F,(n) | ←(C) | ↓ | ↓ | ↓ | 0 | ↓ | PF | 0 | • | 11 | 101 | 101 | ED | 2 | 3 | 12 | 011 | E |
| | | | | | | | | | | 01 | 110 | 000 | 70 | | | | 100 | H |
| INI | (HL)←(C) HL←HL+1 B←B-1 | ↑ ¹ | ↑ ¹ | ↑ ¹ | ↑ ³ | ↑ ¹ | X | ↓ ² | ↑ ³ | 11 | 101 | 101 | ED | 2 | 4 | 16 | 101 | L |
| | | | | | | | | | | 10 | 100 | 010 | A2 | | | | 111 | A |
| INIR | (HL)←(C) HL←HL+1 B←B-1 Repeat until B=0 | 0 | 1 | 0 | ↑ ³ | 0 | X | ↓ ² | ↑ ³ | 11 | 101 | 101 | ED | 2 | 5 | 21 | if B≠0 | |
| | | | | | | | | | | 10 | 110 | 010 | B2 | 2 | 4 | 16 | if B=0 | |
| IND | (HL)←(C) HL←HL-1 B←B-1 | ↑ ¹ | ↑ ¹ | ↑ ¹ | ↑ ⁴ | ↑ ¹ | X | ↓ ² | ↑ ⁴ | 11 | 101 | 101 | ED | 2 | 4 | 16 | | |
| | | | | | | | | | | 10 | 101 | 010 | AA | | | | | |
| INDR | (HL)←(C) HL←HL-1 B←B-1 Repeat until B=0 | 0 | 1 | 0 | ↑ ⁴ | 0 | X | ↓ ² | ↑ ⁴ | 11 | 101 | 101 | ED | 2 | 5 | 21 | if B≠0 | |
| | | | | | | | | | | 10 | 111 | 010 | BA | 2 | 4 | 16 | if B=0 | |
| OUT (n),A | (n)←A | • | • | • | • | • | • | • | • | 11 | 010 | 011 | D3 | 2 | 3 | 11 | | |
| | | | | | | | | | | ← n → | | | | | | | | |
| OUT (C),r | (C)←r | • | • | • | • | • | • | • | • | 11 | 101 | 101 | ED | 2 | 3 | 12 | | |
| | | | | | | | | | | 01 | r | 001 | | | | | | |
| OUT (C),0 | (C)←0 | • | • | • | • | • | • | • | • | 11 | 101 | 101 | ED | 2 | 3 | 12 | | |
| | | | | | | | | | | 01 | 110 | 001 | 71 | | | | | |
| OUTI | (C)←(HL) HL←HL+1 B←B-1 | ↑ ¹ | ↑ ¹ | ↑ ¹ | ↑ ⁵ | ↑ ¹ | X | ↓ ² | ↑ ⁵ | 11 | 101 | 101 | ED | 2 | 4 | 16 | | |
| | | | | | | | | | | 10 | 100 | 011 | A2 | | | | | |
| OTIR | (C)←(HL) HL←HL+1 B←B-1 Repeat until B=0 | 0 | 1 | 0 | ↑ ⁵ | 0 | X | ↓ ² | ↑ ⁵ | 11 | 101 | 101 | ED | 2 | 5 | 21 | if B≠0 | |
| | | | | | | | | | | 10 | 110 | 011 | B3 | 2 | 4 | 16 | if B=0 | |
| OUTD | (C)←(HL) HL←HL-1 B←B-1 | ↑ ¹ | ↑ ¹ | ↑ ¹ | ↑ ⁵ | ↑ ¹ | X | ↓ ² | ↑ ⁵ | 11 | 101 | 101 | ED | 2 | 4 | 16 | | |
| | | | | | | | | | | 10 | 101 | 011 | AA | | | | | |
| OTDR | (C)←(HL) HL←HL-1 B←B-1 Repeat until B=0 | 0 | 1 | 0 | ↑ ⁵ | 0 | X | ↓ ² | ↑ ⁵ | 11 | 101 | 101 | ED | 2 | 5 | 21 | if B≠0 | |
| | | | | | | | | | | 10 | 111 | 011 | BB | 2 | 4 | 16 | if B=0 | |

Note: ¹ flag is affected by the result of B←B-1 as in DEC B.
² NF is a copy of bit 7 of the transferred byte.
³ This flag contains the carry of (((C+1) AND 255)+(C)), see section 4.3.
⁴ This flag contains the carry of (((C-1) AND 255)+(C)), see section 4.3.
⁵ This flag contains the carry of (L+(C)), see section 4.3.

Chapter 9

Instructions Sorted by Opcode

Any instruction marked with * is undocumented.

| | | | | | |
|--------|--------------|--------|--------------|----|------------|
| 00 | NOP | 2D | DEC L | 5A | LD E,D |
| 01 n n | LD BC,nn | 2E n | LD L,n | 5B | LD E,E |
| 02 | LD (BC),A | 2F | CPL | 5C | LD E,H |
| 03 | INC BC | 30 e | JR NC,(PC+e) | 5D | LD E,L |
| 04 | INC B | 31 n n | LD SP,nn | 5E | LD E,(HL) |
| 05 | DEC B | 32 n n | LD (nn),A | 5F | LD E,A |
| 06 n | LD B,n | 33 | INC SP | 60 | LD H,B |
| 07 | RLCA | 34 | INC (HL) | 61 | LD H,C |
| 08 | EX AF,AF' | 35 | DEC (HL) | 62 | LD H,D |
| 09 | ADD HL,BC | 36 n | LD (HL),n | 63 | LD H,E |
| 0A | LD A,(BC) | 37 | SCF | 64 | LD H,H |
| 0B | DEC BC | 38 e | JR C,(PC+e) | 65 | LD H,L |
| 0C | INC C | 39 | ADD HL,SP | 66 | LD H,(HL) |
| 0D | DEC C | 3A n n | LD A,(nn) | 67 | LD H,A |
| 0E n | LD C,n | 3B | DEC SP | 68 | LD L,B |
| 0F | RRCA | 3C | INC A | 69 | LD L,C |
| 10 e | DJNZ (PC+e) | 3D | DEC A | 6A | LD L,D |
| 11 n n | LD DE,nn | 3E n | LD A,n | 6B | LD L,E |
| 12 | LD (DE),A | 3F | CCF | 6C | LD L,H |
| 13 | INC DE | 40 | LD B,B | 6D | LD L,L |
| 14 | INC D | 41 | LD B,C | 6E | LD L,(HL) |
| 15 | DEC D | 42 | LD B,D | 6F | LD L,A |
| 16 n | LD D,n | 43 | LD B,E | 70 | LD (HL),B |
| 17 | RLA | 44 | LD B,H | 71 | LD (HL),C |
| 18 e | JR (PC+e) | 45 | LD B,L | 72 | LD (HL),D |
| 19 | ADD HL,DE | 46 | LD B,(HL) | 73 | LD (HL),E |
| 1A | LD A,(DE) | 47 | LD B,A | 74 | LD (HL),H |
| 1B | DEC DE | 48 | LD C,B | 75 | LD (HL),L |
| 1C | INC E | 49 | LD C,C | 76 | HALT |
| 1D | DEC E | 4A | LD C,D | 77 | LD (HL),A |
| 1E n | LD E,n | 4B | LD C,E | 78 | LD A,B |
| 1F | RRA | 4C | LD C,H | 79 | LD A,C |
| 20 e | JR NZ,(PC+e) | 4D | LD C,L | 7A | LD A,D |
| 21 n n | LD HL,nn | 4E | LD C,(HL) | 7B | LD A,E |
| 22 n n | LD (nn),HL | 4F | LD C,A | 7C | LD A,H |
| 23 | INC HL | 50 | LD D,B | 7D | LD A,L |
| 24 | INC H | 51 | LD D,C | 7E | LD A,(HL) |
| 25 | DEC H | 52 | LD D,D | 7F | LD A,A |
| 26 n | LD H,n | 53 | LD D,E | 80 | ADD A,B |
| 27 | DAA | 54 | LD D,H | 81 | ADD A,C |
| 28 e | JR Z,(PC+e) | 55 | LD D,L | 82 | ADD A,D |
| 29 | ADD HL,HL | 56 | LD D,(HL) | 83 | ADD A,E |
| 2A n n | LD HL,(nn) | 57 | LD D,A | 84 | ADD A,H |
| 2B | DEC HL | 58 | LD E,B | 85 | ADD A,L |
| 2C | INC L | 59 | LD E,C | 86 | ADD A,(HL) |

CHAPTER 9. INSTRUCTIONS SORTED BY OPCODE

| | | | | | |
|--------|--------------|------|------------|------|------------|
| 87 | ADD A,A | CB06 | RLC (HL) | CB50 | BIT 2,B |
| 88 | ADC A,B | CB07 | RLC A | CB51 | BIT 2,C |
| 89 | ADC A,C | CB08 | RRC B | CB52 | BIT 2,D |
| 8A | ADC A,D | CB09 | RRC C | CB53 | BIT 2,E |
| 8B | ADC A,E | CB0A | RRC D | CB54 | BIT 2,H |
| 8C | ADC A,H | CB0B | RRC E | CB55 | BIT 2,L |
| 8D | ADC A,L | CB0C | RRC H | CB56 | BIT 2,(HL) |
| 8E | ADC A,(HL) | CB0D | RRC L | CB57 | BIT 2,A |
| 8F | ADC A,A | CB0E | RRC (HL) | CB58 | BIT 3,B |
| 90 | SUB B | CB0F | RRC A | CB59 | BIT 3,C |
| 91 | SUB C | CB10 | RL B | CB5A | BIT 3,D |
| 92 | SUB D | CB11 | RL C | CB5B | BIT 3,E |
| 93 | SUB E | CB12 | RL D | CB5C | BIT 3,H |
| 94 | SUB H | CB13 | RL E | CB5D | BIT 3,L |
| 95 | SUB L | CB14 | RL H | CB5E | BIT 3,(HL) |
| 96 | SUB (HL) | CB15 | RL L | CB5F | BIT 3,A |
| 97 | SUB A | CB16 | RL (HL) | CB60 | BIT 4,B |
| 98 | SBC A,B | CB17 | RL A | CB61 | BIT 4,C |
| 99 | SBC A,C | CB18 | RR B | CB62 | BIT 4,D |
| 9A | SBC A,D | CB19 | RR C | CB63 | BIT 4,E |
| 9B | SBC A,E | CB1A | RR D | CB64 | BIT 4,H |
| 9C | SBC A,H | CB1B | RR E | CB65 | BIT 4,L |
| 9D | SBC A,L | CB1C | RR H | CB66 | BIT 4,(HL) |
| 9E | SBC A,(HL) | CB1D | RR L | CB67 | BIT 4,A |
| 9F | SBC A,A | CB1E | RR (HL) | CB68 | BIT 5,B |
| A0 | AND B | CB1F | RR A | CB69 | BIT 5,C |
| A1 | AND C | CB20 | SLA B | CB6A | BIT 5,D |
| A2 | AND D | CB21 | SLA C | CB6B | BIT 5,E |
| A3 | AND E | CB22 | SLA D | CB6C | BIT 5,H |
| A4 | AND H | CB23 | SLA E | CB6D | BIT 5,L |
| A5 | AND L | CB24 | SLA H | CB6E | BIT 5,(HL) |
| A6 | AND (HL) | CB25 | SLA L | CB6F | BIT 5,A |
| A7 | AND A | CB26 | SLA (HL) | CB70 | BIT 6,B |
| A8 | XOR B | CB27 | SLA A | CB71 | BIT 6,C |
| A9 | XOR C | CB28 | SRA B | CB72 | BIT 6,D |
| AA | XOR D | CB29 | SRA C | CB73 | BIT 6,E |
| AB | XOR E | CB2A | SRA D | CB74 | BIT 6,H |
| AC | XOR H | CB2B | SRA E | CB75 | BIT 6,L |
| AD | XOR L | CB2C | SRA H | CB76 | BIT 6,(HL) |
| AE | XOR (HL) | CB2D | SRA L | CB77 | BIT 6,A |
| AF | XOR A | CB2E | SRA (HL) | CB78 | BIT 7,B |
| B0 | OR B | CB2F | SRA A | CB79 | BIT 7,C |
| B1 | OR C | CB30 | SLL B* | CB7A | BIT 7,D |
| B2 | OR D | CB31 | SLL C* | CB7B | BIT 7,E |
| B3 | OR E | CB32 | SLL D* | CB7C | BIT 7,H |
| B4 | OR H | CB33 | SLL E* | CB7D | BIT 7,L |
| B5 | OR L | CB34 | SLL H* | CB7E | BIT 7,(HL) |
| B6 | OR (HL) | CB35 | SLL L* | CB7F | BIT 7,A |
| B7 | OR A | CB36 | SLL (HL)* | CB80 | RES 0,B |
| B8 | CP B | CB37 | SLL A* | CB81 | RES 0,C |
| B9 | CP C | CB38 | SRL B | CB82 | RES 0,D |
| BA | CP D | CB39 | SRL C | CB83 | RES 0,E |
| BB | CP E | CB3A | SRL D | CB84 | RES 0,H |
| BC | CP H | CB3B | SRL E | CB85 | RES 0,L |
| BD | CP L | CB3C | SRL H | CB86 | RES 0,(HL) |
| BE | CP (HL) | CB3D | SRL L | CB87 | RES 0,A |
| BF | CP A | CB3E | SRL (HL) | CB88 | RES 1,B |
| C0 | RET NZ | CB3F | SRL A | CB89 | RES 1,C |
| C1 | POP BC | CB40 | BIT 0,B | CB8A | RES 1,D |
| C2 n n | JP NZ,(nn) | CB41 | BIT 0,C | CB8B | RES 1,E |
| C3 n n | JP (nn) | CB42 | BIT 0,D | CB8C | RES 1,H |
| C4 n n | CALL NZ,(nn) | CB43 | BIT 0,E | CB8D | RES 1,L |
| C5 | PUSH BC | CB44 | BIT 0,H | CB8E | RES 1,(HL) |
| C6 n | ADD A,n | CB45 | BIT 0,L | CB8F | RES 1,A |
| C7 | RST OH | CB46 | BIT 0,(HL) | CB90 | RES 2,B |
| C8 | RET Z | CB47 | BIT 0,A | CB91 | RES 2,C |
| C9 | RET | CB48 | BIT 1,B | CB92 | RES 2,D |
| CA n n | JP Z,(nn) | CB49 | BIT 1,C | CB93 | RES 2,E |
| CB00 | RLC B | CB4A | BIT 1,D | CB94 | RES 2,H |
| CB01 | RLC C | CB4B | BIT 1,E | CB95 | RES 2,L |
| CB02 | RLC D | CB4C | BIT 1,H | CB96 | RES 2,(HL) |
| CB03 | RLC E | CB4D | BIT 1,L | CB97 | RES 2,A |
| CB04 | RLC H | CB4E | BIT 1,(HL) | CB98 | RES 3,B |
| CB05 | RLC L | CB4F | BIT 1,A | CB99 | RES 3,C |

CHAPTER 9. INSTRUCTIONS SORTED BY OPCODE

| | | | | | |
|------|------------|----------|--------------|-----------|---------------|
| CB9A | RES 3,D | CBE4 | SET 4,H | DD5E d | LD E,(IX+d) |
| CB9B | RES 3,E | CBE5 | SET 4,L | DD60 | LD IXh,B* |
| CB9C | RES 3,H | CBE6 | SET 4,(HL) | DD61 | LD IXh,C* |
| CB9D | RES 3,L | CBE7 | SET 4,A | DD62 | LD IXh,D* |
| CB9E | RES 3,(HL) | CBE8 | SET 5,B | DD63 | LD IXh,E* |
| CB9F | RES 3,A | CBE9 | SET 5,C | DD64 | LD IXh,IXh* |
| CBA0 | RES 4,B | CBEA | SET 5,D | DD65 | LD IXh,IXl* |
| CBA1 | RES 4,C | CBEB | SET 5,E | DD66 d | LD H,(IX+d) |
| CBA2 | RES 4,D | CBEC | SET 5,H | DD67 | LD IXh,A* |
| CBA3 | RES 4,E | CBED | SET 5,L | DD68 | LD IXl,B* |
| CBA4 | RES 4,H | CBEE | SET 5,(HL) | DD69 | LD IXl,C* |
| CBA5 | RES 4,L | CBEF | SET 5,A | DD6A | LD IXl,D* |
| CBA6 | RES 4,(HL) | CBFO | SET 6,B | DD6B | LD IXl,E* |
| CBA7 | RES 4,A | CBF1 | SET 6,C | DD6C | LD IXl,IXh* |
| CBA8 | RES 5,B | CBF2 | SET 6,D | DD6D | LD IXl,IXl* |
| CBA9 | RES 5,C | CBF3 | SET 6,E | DD6E d | LD L,(IX+d) |
| CBAA | RES 5,D | CBF4 | SET 6,H | DD6F | LD IXl,A* |
| CBAB | RES 5,E | CBF5 | SET 6,L | DD70 d | LD (IX+d),B |
| CBAC | RES 5,H | CBF6 | SET 6,(HL) | DD71 d | LD (IX+d),C |
| CBAD | RES 5,L | CBF7 | SET 6,A | DD72 d | LD (IX+d),D |
| CBAE | RES 5,(HL) | CBF8 | SET 7,B | DD73 d | LD (IX+d),E |
| CBAF | RES 5,A | CBF9 | SET 7,C | DD74 d | LD (IX+d),H |
| CBBO | RES 6,B | CBFA | SET 7,D | DD75 d | LD (IX+d),L |
| CBB1 | RES 6,C | CBFB | SET 7,E | DD77 d | LD (IX+d),A |
| CBB2 | RES 6,D | CBFC | SET 7,H | DD7C | LD A,IXh* |
| CBB3 | RES 6,E | CBFD | SET 7,L | DD7D | LD A,IXl* |
| CBB4 | RES 6,H | CBFE | SET 7,(HL) | DD7E d | LD A,(IX+d) |
| CBB5 | RES 6,L | CBFF | SET 7,A | DD84 | ADD A,IXh* |
| CBB6 | RES 6,(HL) | CC n n | CALL Z,(nn) | DD85 | ADD A,IXl* |
| CBB7 | RES 6,A | CD n n | CALL (nn) | DD86 d | ADD A,(IX+d) |
| CBB8 | RES 7,B | CE n | ADC A,n | DD8C | ADC A,IXh* |
| CBB9 | RES 7,C | CF | RST 8H | DD8D | ADC A,IXl* |
| CBBA | RES 7,D | DO | RET NC | DD8E d | ADC A,(IX+d) |
| CBBB | RES 7,E | D1 | POP DE | DD94 | SUB IXh* |
| CBBC | RES 7,H | D2 n n | JP NC,(nn) | DD95 | SUB IXl* |
| CBBD | RES 7,L | D3 n | OUT (n),A | DD96 d | SUB (IX+d) |
| CBBE | RES 7,(HL) | D4 n n | CALL NC,(nn) | DD9C | SBC A,IXh* |
| CBBF | RES 7,A | D5 | PUSH DE | DD9D | SBC A,IXl* |
| CBC0 | SET 0,B | D6 n | SUB n | DD9E d | SBC A,(IX+d) |
| CBC1 | SET 0,C | D7 | RST 10H | DDA4 | AND IXh* |
| CBC2 | SET 0,D | D8 | RET C | DDA5 | AND IXl* |
| CBC3 | SET 0,E | D9 | EXX | DDA6 d | AND (IX+d) |
| CBC4 | SET 0,H | DA n n | JP C,(nn) | DDAC | XOR IXh* |
| CBC5 | SET 0,L | DB n | IN A,(n) | DDAD | XOR IXl* |
| CBC6 | SET 0,(HL) | DC n n | CALL C,(nn) | DDAE d | XOR (IX+d) |
| CBC7 | SET 0,A | DD09 | ADD IX,BC | DDB4 | OR IXh* |
| CBC8 | SET 1,B | DD19 | ADD IX,DE | DDB5 | OR IXl* |
| CBC9 | SET 1,C | DD21 n n | LD IX,nn | DDB6 d | OR (IX+d) |
| CBCA | SET 1,D | DD22 n n | LD (nn),IX | DDBC | CP IXh* |
| CBCB | SET 1,E | DD23 | INC IX | DDBD | CP IXl* |
| CBCC | SET 1,H | DD24 | INC IXh* | DDBE d | CP (IX+d) |
| CBCD | SET 1,L | DD25 | DEC IXh* | DDCB d 00 | RLC (IX+d),B* |
| CBCE | SET 1,(HL) | DD26 n | LD IXh,n* | DDCB d 01 | RLC (IX+d),C* |
| CBCF | SET 1,A | DD29 | ADD IX,IX | DDCB d 02 | RLC (IX+d),D* |
| CBD0 | SET 2,B | DD2A n n | LD IX,(nn) | DDCB d 03 | RLC (IX+d),E* |
| CBD1 | SET 2,C | DD2B | DEC IX | DDCB d 04 | RLC (IX+d),H* |
| CBD2 | SET 2,D | DD2C | INC IXl* | DDCB d 05 | RLC (IX+d),L* |
| CBD3 | SET 2,E | DD2D | DEC IXl* | DDCB d 06 | RLC (IX+d) |
| CBD4 | SET 2,H | DD2E n | LD IXl,n* | DDCB d 07 | RLC (IX+d),A* |
| CBD5 | SET 2,L | DD34 d | INC (IX+d) | DDCB d 08 | RRC (IX+d),B* |
| CBD6 | SET 2,(HL) | DD35 d | DEC (IX+d) | DDCB d 09 | RRC (IX+d),C* |
| CBD7 | SET 2,A | DD36 d n | LD (IX+d),n | DDCB d 0A | RRC (IX+d),D* |
| CBD8 | SET 3,B | DD39 | ADD IX,SP | DDCB d 0B | RRC (IX+d),E* |
| CBD9 | SET 3,C | DD44 | LD B,IXh* | DDCB d 0C | RRC (IX+d),H* |
| CBDA | SET 3,D | DD45 | LD B,IXl* | DDCB d 0D | RRC (IX+d),L* |
| CBDB | SET 3,E | DD46 d | LD B,(IX+d) | DDCB d 0E | RRC (IX+d) |
| CBDC | SET 3,H | DD4C | LD C,IXh* | DDCB d 0F | RRC (IX+d),A* |
| CBDD | SET 3,L | DD4D | LD C,IXl* | DDCB d 10 | RL (IX+d),B* |
| CBDE | SET 3,(HL) | DD4E d | LD C,(IX+d) | DDCB d 11 | RL (IX+d),C* |
| CBDF | SET 3,A | DD54 | LD D,IXh* | DDCB d 12 | RL (IX+d),D* |
| CBE0 | SET 4,B | DD55 | LD D,IXl* | DDCB d 13 | RL (IX+d),E* |
| CBE1 | SET 4,C | DD56 d | LD D,(IX+d) | DDCB d 14 | RL (IX+d),H* |
| CBE2 | SET 4,D | DD5C | LD E,IXh* | DDCB d 15 | RL (IX+d),L* |
| CBE3 | SET 4,E | DD5D | LD E,IXl* | DDCB d 16 | RL (IX+d) |

CHAPTER 9. INSTRUCTIONS SORTED BY OPCODE

| | | | | | |
|-----------|---------------|-----------|-----------------|-----------|-----------------|
| DDCB d 17 | RL (IX+d),A* | DDCB d 61 | BIT 4,(IX+d)* | DDCB d AB | RES 5,(IX+d),E* |
| DDCB d 18 | RR (IX+d),B* | DDCB d 62 | BIT 4,(IX+d)* | DDCB d AC | RES 5,(IX+d),H* |
| DDCB d 19 | RR (IX+d),C* | DDCB d 63 | BIT 4,(IX+d)* | DDCB d AD | RES 5,(IX+d),L* |
| DDCB d 1A | RR (IX+d),D* | DDCB d 64 | BIT 4,(IX+d)* | DDCB d AE | RES 5,(IX+d) |
| DDCB d 1B | RR (IX+d),E* | DDCB d 65 | BIT 4,(IX+d)* | DDCB d AF | RES 5,(IX+d),A* |
| DDCB d 1C | RR (IX+d),H* | DDCB d 66 | BIT 4,(IX+d) | DDCB d B0 | RES 6,(IX+d),B* |
| DDCB d 1D | RR (IX+d),L* | DDCB d 67 | BIT 4,(IX+d)* | DDCB d B1 | RES 6,(IX+d),C* |
| DDCB d 1E | RR (IX+d) | DDCB d 68 | BIT 5,(IX+d)* | DDCB d B2 | RES 6,(IX+d),D* |
| DDCB d 1F | RR (IX+d),A* | DDCB d 69 | BIT 5,(IX+d)* | DDCB d B3 | RES 6,(IX+d),E* |
| DDCB d 20 | SLA (IX+d),B* | DDCB d 6A | BIT 5,(IX+d)* | DDCB d B4 | RES 6,(IX+d),H* |
| DDCB d 21 | SLA (IX+d),C* | DDCB d 6B | BIT 5,(IX+d)* | DDCB d B5 | RES 6,(IX+d),L* |
| DDCB d 22 | SLA (IX+d),D* | DDCB d 6C | BIT 5,(IX+d)* | DDCB d B6 | RES 6,(IX+d) |
| DDCB d 23 | SLA (IX+d),E* | DDCB d 6D | BIT 5,(IX+d)* | DDCB d B7 | RES 6,(IX+d),A* |
| DDCB d 24 | SLA (IX+d),H* | DDCB d 6E | BIT 5,(IX+d) | DDCB d B8 | RES 7,(IX+d),B* |
| DDCB d 25 | SLA (IX+d),L* | DDCB d 6F | BIT 5,(IX+d)* | DDCB d B9 | RES 7,(IX+d),C* |
| DDCB d 26 | SLA (IX+d) | DDCB d 70 | BIT 6,(IX+d)* | DDCB d BA | RES 7,(IX+d),D* |
| DDCB d 27 | SLA (IX+d),A* | DDCB d 71 | BIT 6,(IX+d)* | DDCB d BB | RES 7,(IX+d),E* |
| DDCB d 28 | SRA (IX+d),B* | DDCB d 72 | BIT 6,(IX+d)* | DDCB d BC | RES 7,(IX+d),H* |
| DDCB d 29 | SRA (IX+d),C* | DDCB d 73 | BIT 6,(IX+d)* | DDCB d BD | RES 7,(IX+d),L* |
| DDCB d 2A | SRA (IX+d),D* | DDCB d 74 | BIT 6,(IX+d)* | DDCB d BE | RES 7,(IX+d) |
| DDCB d 2B | SRA (IX+d),E* | DDCB d 75 | BIT 6,(IX+d)* | DDCB d BF | RES 7,(IX+d),A* |
| DDCB d 2C | SRA (IX+d),H* | DDCB d 76 | BIT 6,(IX+d) | DDCB d C0 | SET 0,(IX+d),B* |
| DDCB d 2D | SRA (IX+d),L* | DDCB d 77 | BIT 6,(IX+d)* | DDCB d C1 | SET 0,(IX+d),C* |
| DDCB d 2E | SRA (IX+d) | DDCB d 78 | BIT 7,(IX+d)* | DDCB d C2 | SET 0,(IX+d),D* |
| DDCB d 2F | SRA (IX+d),A* | DDCB d 79 | BIT 7,(IX+d)* | DDCB d C3 | SET 0,(IX+d),E* |
| DDCB d 30 | SLL (IX+d),B* | DDCB d 7A | BIT 7,(IX+d)* | DDCB d C4 | SET 0,(IX+d),H* |
| DDCB d 31 | SLL (IX+d),C* | DDCB d 7B | BIT 7,(IX+d)* | DDCB d C5 | SET 0,(IX+d),L* |
| DDCB d 32 | SLL (IX+d),D* | DDCB d 7C | BIT 7,(IX+d)* | DDCB d C6 | SET 0,(IX+d) |
| DDCB d 33 | SLL (IX+d),E* | DDCB d 7D | BIT 7,(IX+d)* | DDCB d C7 | SET 0,(IX+d),A* |
| DDCB d 34 | SLL (IX+d),H* | DDCB d 7E | BIT 7,(IX+d) | DDCB d C8 | SET 1,(IX+d),B* |
| DDCB d 35 | SLL (IX+d),L* | DDCB d 7F | BIT 7,(IX+d)* | DDCB d C9 | SET 1,(IX+d),C* |
| DDCB d 36 | SLL (IX+d)* | DDCB d 80 | RES 0,(IX+d),B* | DDCB d CA | SET 1,(IX+d),D* |
| DDCB d 37 | SLL (IX+d),A* | DDCB d 81 | RES 0,(IX+d),C* | DDCB d CB | SET 1,(IX+d),E* |
| DDCB d 38 | SRL (IX+d),B* | DDCB d 82 | RES 0,(IX+d),D* | DDCB d CC | SET 1,(IX+d),H* |
| DDCB d 39 | SRL (IX+d),C* | DDCB d 83 | RES 0,(IX+d),E* | DDCB d CD | SET 1,(IX+d),L* |
| DDCB d 3A | SRL (IX+d),D* | DDCB d 84 | RES 0,(IX+d),H* | DDCB d CE | SET 1,(IX+d) |
| DDCB d 3B | SRL (IX+d),E* | DDCB d 85 | RES 0,(IX+d),L* | DDCB d CF | SET 1,(IX+d),A* |
| DDCB d 3C | SRL (IX+d),H* | DDCB d 86 | RES 0,(IX+d) | DDCB d D0 | SET 2,(IX+d),B* |
| DDCB d 3D | SRL (IX+d),L* | DDCB d 87 | RES 0,(IX+d),A* | DDCB d D1 | SET 2,(IX+d),C* |
| DDCB d 3E | SRL (IX+d) | DDCB d 88 | RES 1,(IX+d),B* | DDCB d D2 | SET 2,(IX+d),D* |
| DDCB d 3F | SRL (IX+d),A* | DDCB d 89 | RES 1,(IX+d),C* | DDCB d D3 | SET 2,(IX+d),E* |
| DDCB d 40 | BIT 0,(IX+d)* | DDCB d 8A | RES 1,(IX+d),D* | DDCB d D4 | SET 2,(IX+d),H* |
| DDCB d 41 | BIT 0,(IX+d)* | DDCB d 8B | RES 1,(IX+d),E* | DDCB d D5 | SET 2,(IX+d),L* |
| DDCB d 42 | BIT 0,(IX+d)* | DDCB d 8C | RES 1,(IX+d),H* | DDCB d D6 | SET 2,(IX+d) |
| DDCB d 43 | BIT 0,(IX+d)* | DDCB d 8D | RES 1,(IX+d),L* | DDCB d D7 | SET 2,(IX+d),A* |
| DDCB d 44 | BIT 0,(IX+d)* | DDCB d 8E | RES 1,(IX+d) | DDCB d D8 | SET 3,(IX+d),B* |
| DDCB d 45 | BIT 0,(IX+d)* | DDCB d 8F | RES 1,(IX+d),A* | DDCB d D9 | SET 3,(IX+d),C* |
| DDCB d 46 | BIT 0,(IX+d) | DDCB d 90 | RES 2,(IX+d),B* | DDCB d DA | SET 3,(IX+d),D* |
| DDCB d 47 | BIT 0,(IX+d)* | DDCB d 91 | RES 2,(IX+d),C* | DDCB d DB | SET 3,(IX+d),E* |
| DDCB d 48 | BIT 1,(IX+d)* | DDCB d 92 | RES 2,(IX+d),D* | DDCB d DC | SET 3,(IX+d),H* |
| DDCB d 49 | BIT 1,(IX+d)* | DDCB d 93 | RES 2,(IX+d),E* | DDCB d DD | SET 3,(IX+d),L* |
| DDCB d 4A | BIT 1,(IX+d)* | DDCB d 94 | RES 2,(IX+d),H* | DDCB d DE | SET 3,(IX+d) |
| DDCB d 4B | BIT 1,(IX+d)* | DDCB d 95 | RES 2,(IX+d),L* | DDCB d DF | SET 3,(IX+d),A* |
| DDCB d 4C | BIT 1,(IX+d)* | DDCB d 96 | RES 2,(IX+d) | DDCB d E0 | SET 4,(IX+d),B* |
| DDCB d 4D | BIT 1,(IX+d)* | DDCB d 97 | RES 2,(IX+d),A* | DDCB d E1 | SET 4,(IX+d),C* |
| DDCB d 4E | BIT 1,(IX+d) | DDCB d 98 | RES 3,(IX+d),B* | DDCB d E2 | SET 4,(IX+d),D* |
| DDCB d 4F | BIT 1,(IX+d)* | DDCB d 99 | RES 3,(IX+d),C* | DDCB d E3 | SET 4,(IX+d),E* |
| DDCB d 50 | BIT 2,(IX+d)* | DDCB d 9A | RES 3,(IX+d),D* | DDCB d E4 | SET 4,(IX+d),H* |
| DDCB d 51 | BIT 2,(IX+d)* | DDCB d 9B | RES 3,(IX+d),E* | DDCB d E5 | SET 4,(IX+d),L* |
| DDCB d 52 | BIT 2,(IX+d)* | DDCB d 9C | RES 3,(IX+d),H* | DDCB d E6 | SET 4,(IX+d) |
| DDCB d 53 | BIT 2,(IX+d)* | DDCB d 9D | RES 3,(IX+d),L* | DDCB d E7 | SET 4,(IX+d),A* |
| DDCB d 54 | BIT 2,(IX+d)* | DDCB d 9E | RES 3,(IX+d) | DDCB d E8 | SET 5,(IX+d),B* |
| DDCB d 55 | BIT 2,(IX+d)* | DDCB d 9F | RES 3,(IX+d),A* | DDCB d E9 | SET 5,(IX+d),C* |
| DDCB d 56 | BIT 2,(IX+d) | DDCB d A0 | RES 4,(IX+d),B* | DDCB d EA | SET 5,(IX+d),D* |
| DDCB d 57 | BIT 2,(IX+d)* | DDCB d A1 | RES 4,(IX+d),C* | DDCB d EB | SET 5,(IX+d),E* |
| DDCB d 58 | BIT 3,(IX+d)* | DDCB d A2 | RES 4,(IX+d),D* | DDCB d EC | SET 5,(IX+d),H* |
| DDCB d 59 | BIT 3,(IX+d)* | DDCB d A3 | RES 4,(IX+d),E* | DDCB d ED | SET 5,(IX+d),L* |
| DDCB d 5A | BIT 3,(IX+d)* | DDCB d A4 | RES 4,(IX+d),H* | DDCB d EE | SET 5,(IX+d) |
| DDCB d 5B | BIT 3,(IX+d)* | DDCB d A5 | RES 4,(IX+d),L* | DDCB d EF | SET 5,(IX+d),A* |
| DDCB d 5C | BIT 3,(IX+d)* | DDCB d A6 | RES 4,(IX+d) | DDCB d F0 | SET 6,(IX+d),B* |
| DDCB d 5D | BIT 3,(IX+d)* | DDCB d A7 | RES 4,(IX+d),A* | DDCB d F1 | SET 6,(IX+d),C* |
| DDCB d 5E | BIT 3,(IX+d) | DDCB d A8 | RES 5,(IX+d),B* | DDCB d F2 | SET 6,(IX+d),D* |
| DDCB d 5F | BIT 3,(IX+d)* | DDCB d A9 | RES 5,(IX+d),C* | DDCB d F3 | SET 6,(IX+d),E* |
| DDCB d 60 | BIT 4,(IX+d)* | DDCB d AA | RES 5,(IX+d),D* | DDCB d F4 | SET 6,(IX+d),H* |

CHAPTER 9. INSTRUCTIONS SORTED BY OPCODE

| | | | | | |
|-----------|-------------------|----------|----------------------|-----------|----------------|
| DDCB d F5 | SET 6, (IX+d), L* | ED6B n n | LD HL, (nn) | FD54 | LD D, IYh* |
| DDCB d F6 | SET 6, (IX+d) | ED6C | NEG* | FD55 | LD D, IY1* |
| DDCB d F7 | SET 6, (IX+d), A* | ED6D | RETn* | FD56 d | LD D, (IY+d) |
| DDCB d F8 | SET 7, (IX+d), B* | ED6E | IM 0* | FD5C | LD E, IYh* |
| DDCB d F9 | SET 7, (IX+d), C* | ED6F | RLD | FD5D | LD E, IY1* |
| DDCB d FA | SET 7, (IX+d), D* | ED70 | IN F, (C)* / IN (C)* | FD5E d | LD E, (IY+d) |
| DDCB d FB | SET 7, (IX+d), E* | ED71 | OUT (C), 0* | FD60 | LD IYh, B* |
| DDCB d FC | SET 7, (IX+d), H* | ED72 | SBC HL, SP | FD61 | LD IYh, C* |
| DDCB d FD | SET 7, (IX+d), L* | ED73 n n | LD (nn), SP | FD62 | LD IYh, D* |
| DDCB d FE | SET 7, (IX+d) | ED74 | NEG* | FD63 | LD IYh, E* |
| DDCB d FF | SET 7, (IX+d), A* | ED75 | RETn* | FD64 | LD IYh, IYh* |
| DDE1 | POP IX | ED76 | IM 1* | FD65 | LD IYh, IY1* |
| DDE3 | EX (SP), IX | ED78 | IN A, (C) | FD66 d | LD H, (IY+d) |
| DDE5 | PUSH IX | ED79 | OUT (C), A | FD67 | LD IYh, A* |
| DDE9 | JP (IX) | ED7A | ADC HL, SP | FD68 | LD IY1, B* |
| DDF9 | LD SP, IX | ED7B n n | LD SP, (nn) | FD69 | LD IY1, C* |
| DE n | SBC A, n | ED7C | NEG* | FD6A | LD IY1, D* |
| DF | RST 18H | ED7D | RETn* | FD6B | LD IY1, E* |
| E0 | RET P0 | ED7E | IM 2* | FD6C | LD IY1, IYh* |
| E1 | POP HL | EDA0 | LDI | FD6D | LD IY1, IY1* |
| E2 n n | JP P0, (nn) | EDA1 | CPI | FD6E d | LD L, (IY+d) |
| E3 | EX (SP), HL | EDA2 | INI | FD6F | LD IY1, A* |
| E4 n n | CALL P0, (nn) | EDA3 | OUTI | FD70 d | LD (IY+d), B |
| E5 | PUSH HL | EDA8 | LDD | FD71 d | LD (IY+d), C |
| E6 n | AND n | EDA9 | CPD | FD72 d | LD (IY+d), D |
| E7 | RST 20H | EDAA | IND | FD73 d | LD (IY+d), E |
| E8 | RET PE | EDAB | OUTD | FD74 d | LD (IY+d), H |
| E9 | JP (HL) | EDB0 | LDIR | FD75 d | LD (IY+d), L |
| EA n n | JP PE, (nn) | EDB1 | CPIR | FD77 d | LD (IY+d), A |
| EB | EX DE, HL | EDB2 | INIR | FD7C | LD A, IYh* |
| EC n n | CALL PE, (nn) | EDB3 | OTIR | FD7D | LD A, IY1* |
| ED40 | IN B, (C) | EDB8 | LDDR | FD7E d | LD A, (IY+d) |
| ED41 | OUT (C), B | EDB9 | CPDR | FD84 | ADD A, IYh* |
| ED42 | SBC HL, BC | EDBA | INDR | FD85 | ADD A, IY1* |
| ED43 n n | LD (nn), BC | EDBB | OTDR | FD86 d | ADD A, (IY+d) |
| ED44 | NEG | EE n | XOR n | FD8C | ADC A, IYh* |
| ED45 | RETn | EF | RST 28H | FD8D | ADC A, IY1* |
| ED46 | IM 0 | F0 | RET P | FD8E d | ADC A, (IY+d) |
| ED47 | LD I, A | F1 | POP AF | FD94 | SUB IYh* |
| ED48 | IN C, (C) | F2 n n | JP P, (nn) | FD95 | SUB IY1* |
| ED49 | OUT (C), C | F3 | DI | FD96 d | SUB (IY+d) |
| ED4A | ADC HL, BC | F4 n n | CALL P, (nn) | FD9C | SBC A, IYh* |
| ED4B n n | LD BC, (nn) | F5 | PUSH AF | FD9D | SBC A, IY1* |
| ED4C | NEG* | F6 n | OR n | FD9E d | SBC A, (IY+d) |
| ED4D | RETI | F7 | RST 30H | FDA4 | AND IYh* |
| ED4E | IM 0* | F8 | RET M | FDA5 | AND IY1* |
| ED4F | LD R, A | F9 | LD SP, HL | FDA6 d | AND (IY+d) |
| ED50 | IN D, (C) | FA n n | JP M, (nn) | FDAC | XOR IYh* |
| ED51 | OUT (C), D | FB | EI | FDAD | XOR IY1* |
| ED52 | SBC HL, DE | FC n n | CALL M, (nn) | FDAE d | XOR (IY+d) |
| ED53 n n | LD (nn), DE | FD09 | ADD IY, BC | FDB4 | OR IYh* |
| ED54 | NEG* | FD19 | ADD IY, DE | FDB5 | OR IY1* |
| ED55 | RETn* | FD21 n n | LD IY, nn | FDB6 d | OR (IY+d) |
| ED56 | IM 1 | FD22 n n | LD (nn), IY | FDBC | CP IYh* |
| ED57 | LD A, I | FD23 | INC IY | FDBD | CP IY1* |
| ED58 | IN E, (C) | FD24 | INC IYh* | FDBE d | CP (IY+d) |
| ED59 | OUT (C), E | FD25 | DEC IYh* | FDCB d 00 | RLC (IY+d), B* |
| ED5A | ADC HL, DE | FD26 n | LD IYh, n* | FDCB d 01 | RLC (IY+d), C* |
| ED5B n n | LD DE, (nn) | FD29 | ADD IY, IY | FDCB d 02 | RLC (IY+d), D* |
| ED5C | NEG* | FD2A n n | LD IY, (nn) | FDCB d 03 | RLC (IY+d), E* |
| ED5D | RETn* | FD2B | DEC IY | FDCB d 04 | RLC (IY+d), H* |
| ED5E | IM 2 | FD2C | INC IY1* | FDCB d 05 | RLC (IY+d), L* |
| ED5F | LD A, R | FD2D | DEC IY1* | FDCB d 06 | RLC (IY+d) |
| ED60 | IN H, (C) | FD2E n | LD IY1, n* | FDCB d 07 | RLC (IY+d), A* |
| ED61 | OUT (C), H | FD34 d | INC (IY+d) | FDCB d 08 | RRC (IY+d), B* |
| ED62 | SBC HL, HL | FD35 d | DEC (IY+d) | FDCB d 09 | RRC (IY+d), C* |
| ED63 n n | LD (nn), HL | FD36 d n | LD (IY+d), n | FDCB d 0A | RRC (IY+d), D* |
| ED64 | NEG* | FD39 | ADD IY, SP | FDCB d 0B | RRC (IY+d), E* |
| ED65 | RETn* | FD44 | LD B, IYh* | FDCB d 0C | RRC (IY+d), H* |
| ED66 | IM 0* | FD45 | LD B, IY1* | FDCB d 0D | RRC (IY+d), L* |
| ED67 | RRD | FD46 d | LD B, (IY+d) | FDCB d 0E | RRC (IY+d) |
| ED68 | IN L, (C) | FD4C | LD C, IYh* | FDCB d 0F | RRC (IY+d), A* |
| ED69 | OUT (C), L | FD4D | LD C, IY1* | FDCB d 10 | RL (IY+d), B* |
| ED6A | ADC HL, HL | FD4E d | LD C, (IY+d) | FDCB d 11 | RL (IY+d), C* |

CHAPTER 9. INSTRUCTIONS SORTED BY OPCODE

| | | | | | |
|-----------|---------------|-----------|-----------------|-----------|-----------------|
| FDCB d 12 | RL (IY+d),D* | FDCB d 5C | BIT 3,(IY+d)* | FDCB d A6 | RES 4,(IY+d) |
| FDCB d 13 | RL (IY+d),E* | FDCB d 5D | BIT 3,(IY+d)* | FDCB d A7 | RES 4,(IY+d),A* |
| FDCB d 14 | RL (IY+d),H* | FDCB d 5E | BIT 3,(IY+d) | FDCB d A8 | RES 5,(IY+d),B* |
| FDCB d 15 | RL (IY+d),L* | FDCB d 5F | BIT 3,(IY+d)* | FDCB d A9 | RES 5,(IY+d),C* |
| FDCB d 16 | RL (IY+d) | FDCB d 60 | BIT 4,(IY+d)* | FDCB d AA | RES 5,(IY+d),D* |
| FDCB d 17 | RL (IY+d),A* | FDCB d 61 | BIT 4,(IY+d)* | FDCB d AB | RES 5,(IY+d),E* |
| FDCB d 18 | RR (IY+d),B* | FDCB d 62 | BIT 4,(IY+d)* | FDCB d AC | RES 5,(IY+d),H* |
| FDCB d 19 | RR (IY+d),C* | FDCB d 63 | BIT 4,(IY+d)* | FDCB d AD | RES 5,(IY+d),L* |
| FDCB d 1A | RR (IY+d),D* | FDCB d 64 | BIT 4,(IY+d)* | FDCB d AE | RES 5,(IY+d) |
| FDCB d 1B | RR (IY+d),E* | FDCB d 65 | BIT 4,(IY+d)* | FDCB d AF | RES 5,(IY+d),A* |
| FDCB d 1C | RR (IY+d),H* | FDCB d 66 | BIT 4,(IY+d) | FDCB d B0 | RES 6,(IY+d),B* |
| FDCB d 1D | RR (IY+d),L* | FDCB d 67 | BIT 4,(IY+d)* | FDCB d B1 | RES 6,(IY+d),C* |
| FDCB d 1E | RR (IY+d) | FDCB d 68 | BIT 5,(IY+d)* | FDCB d B2 | RES 6,(IY+d),D* |
| FDCB d 1F | RR (IY+d),A* | FDCB d 69 | BIT 5,(IY+d)* | FDCB d B3 | RES 6,(IY+d),E* |
| FDCB d 20 | SLA (IY+d),B* | FDCB d 6A | BIT 5,(IY+d)* | FDCB d B4 | RES 6,(IY+d),H* |
| FDCB d 21 | SLA (IY+d),C* | FDCB d 6B | BIT 5,(IY+d)* | FDCB d B5 | RES 6,(IY+d),L* |
| FDCB d 22 | SLA (IY+d),D* | FDCB d 6C | BIT 5,(IY+d)* | FDCB d B6 | RES 6,(IY+d) |
| FDCB d 23 | SLA (IY+d),E* | FDCB d 6D | BIT 5,(IY+d)* | FDCB d B7 | RES 6,(IY+d),A* |
| FDCB d 24 | SLA (IY+d),H* | FDCB d 6E | BIT 5,(IY+d) | FDCB d B8 | RES 7,(IY+d),B* |
| FDCB d 25 | SLA (IY+d),L* | FDCB d 6F | BIT 5,(IY+d)* | FDCB d B9 | RES 7,(IY+d),C* |
| FDCB d 26 | SLA (IY+d) | FDCB d 70 | BIT 6,(IY+d)* | FDCB d BA | RES 7,(IY+d),D* |
| FDCB d 27 | SLA (IY+d),A* | FDCB d 71 | BIT 6,(IY+d)* | FDCB d BB | RES 7,(IY+d),E* |
| FDCB d 28 | SRA (IY+d),B* | FDCB d 72 | BIT 6,(IY+d)* | FDCB d BC | RES 7,(IY+d),H* |
| FDCB d 29 | SRA (IY+d),C* | FDCB d 73 | BIT 6,(IY+d)* | FDCB d BD | RES 7,(IY+d),L* |
| FDCB d 2A | SRA (IY+d),D* | FDCB d 74 | BIT 6,(IY+d)* | FDCB d BE | RES 7,(IY+d) |
| FDCB d 2B | SRA (IY+d),E* | FDCB d 75 | BIT 6,(IY+d)* | FDCB d BF | RES 7,(IY+d),A* |
| FDCB d 2C | SRA (IY+d),H* | FDCB d 76 | BIT 6,(IY+d) | FDCB d C0 | SET 0,(IY+d),B* |
| FDCB d 2D | SRA (IY+d),L* | FDCB d 77 | BIT 6,(IY+d)* | FDCB d C1 | SET 0,(IY+d),C* |
| FDCB d 2E | SRA (IY+d) | FDCB d 78 | BIT 7,(IY+d)* | FDCB d C2 | SET 0,(IY+d),D* |
| FDCB d 2F | SRA (IY+d),A* | FDCB d 79 | BIT 7,(IY+d)* | FDCB d C3 | SET 0,(IY+d),E* |
| FDCB d 30 | SLL (IY+d),B* | FDCB d 7A | BIT 7,(IY+d)* | FDCB d C4 | SET 0,(IY+d),H* |
| FDCB d 31 | SLL (IY+d),C* | FDCB d 7B | BIT 7,(IY+d)* | FDCB d C5 | SET 0,(IY+d),L* |
| FDCB d 32 | SLL (IY+d),D* | FDCB d 7C | BIT 7,(IY+d)* | FDCB d C6 | SET 0,(IY+d) |
| FDCB d 33 | SLL (IY+d),E* | FDCB d 7D | BIT 7,(IY+d)* | FDCB d C7 | SET 0,(IY+d),A* |
| FDCB d 34 | SLL (IY+d),H* | FDCB d 7E | BIT 7,(IY+d) | FDCB d C8 | SET 1,(IY+d),B* |
| FDCB d 35 | SLL (IY+d),L* | FDCB d 7F | BIT 7,(IY+d)* | FDCB d C9 | SET 1,(IY+d),C* |
| FDCB d 36 | SLL (IY+d)* | FDCB d 80 | RES 0,(IY+d),B* | FDCB d CA | SET 1,(IY+d),D* |
| FDCB d 37 | SLL (IY+d),A* | FDCB d 81 | RES 0,(IY+d),C* | FDCB d CB | SET 1,(IY+d),E* |
| FDCB d 38 | SRL (IY+d),B* | FDCB d 82 | RES 0,(IY+d),D* | FDCB d CC | SET 1,(IY+d),H* |
| FDCB d 39 | SRL (IY+d),C* | FDCB d 83 | RES 0,(IY+d),E* | FDCB d CD | SET 1,(IY+d),L* |
| FDCB d 3A | SRL (IY+d),D* | FDCB d 84 | RES 0,(IY+d),H* | FDCB d CE | SET 1,(IY+d) |
| FDCB d 3B | SRL (IY+d),E* | FDCB d 85 | RES 0,(IY+d),L* | FDCB d CF | SET 1,(IY+d),A* |
| FDCB d 3C | SRL (IY+d),H* | FDCB d 86 | RES 0,(IY+d) | FDCB d D0 | SET 2,(IY+d),B* |
| FDCB d 3D | SRL (IY+d),L* | FDCB d 87 | RES 0,(IY+d),A* | FDCB d D1 | SET 2,(IY+d),C* |
| FDCB d 3E | SRL (IY+d) | FDCB d 88 | RES 1,(IY+d),B* | FDCB d D2 | SET 2,(IY+d),D* |
| FDCB d 3F | SRL (IY+d),A* | FDCB d 89 | RES 1,(IY+d),C* | FDCB d D3 | SET 2,(IY+d),E* |
| FDCB d 40 | BIT 0,(IY+d)* | FDCB d 8A | RES 1,(IY+d),D* | FDCB d D4 | SET 2,(IY+d),H* |
| FDCB d 41 | BIT 0,(IY+d)* | FDCB d 8B | RES 1,(IY+d),E* | FDCB d D5 | SET 2,(IY+d),L* |
| FDCB d 42 | BIT 0,(IY+d)* | FDCB d 8C | RES 1,(IY+d),H* | FDCB d D6 | SET 2,(IY+d) |
| FDCB d 43 | BIT 0,(IY+d)* | FDCB d 8D | RES 1,(IY+d),L* | FDCB d D7 | SET 2,(IY+d),A* |
| FDCB d 44 | BIT 0,(IY+d)* | FDCB d 8E | RES 1,(IY+d) | FDCB d D8 | SET 3,(IY+d),B* |
| FDCB d 45 | BIT 0,(IY+d)* | FDCB d 8F | RES 1,(IY+d),A* | FDCB d D9 | SET 3,(IY+d),C* |
| FDCB d 46 | BIT 0,(IY+d) | FDCB d 90 | RES 2,(IY+d),B* | FDCB d DA | SET 3,(IY+d),D* |
| FDCB d 47 | BIT 0,(IY+d)* | FDCB d 91 | RES 2,(IY+d),C* | FDCB d DB | SET 3,(IY+d),E* |
| FDCB d 48 | BIT 1,(IY+d)* | FDCB d 92 | RES 2,(IY+d),D* | FDCB d DC | SET 3,(IY+d),H* |
| FDCB d 49 | BIT 1,(IY+d)* | FDCB d 93 | RES 2,(IY+d),E* | FDCB d DD | SET 3,(IY+d),L* |
| FDCB d 4A | BIT 1,(IY+d)* | FDCB d 94 | RES 2,(IY+d),H* | FDCB d DE | SET 3,(IY+d) |
| FDCB d 4B | BIT 1,(IY+d)* | FDCB d 95 | RES 2,(IY+d),L* | FDCB d DF | SET 3,(IY+d),A* |
| FDCB d 4C | BIT 1,(IY+d)* | FDCB d 96 | RES 2,(IY+d) | FDCB d E0 | SET 4,(IY+d),B* |
| FDCB d 4D | BIT 1,(IY+d)* | FDCB d 97 | RES 2,(IY+d),A* | FDCB d E1 | SET 4,(IY+d),C* |
| FDCB d 4E | BIT 1,(IY+d) | FDCB d 98 | RES 3,(IY+d),B* | FDCB d E2 | SET 4,(IY+d),D* |
| FDCB d 4F | BIT 1,(IY+d)* | FDCB d 99 | RES 3,(IY+d),C* | FDCB d E3 | SET 4,(IY+d),E* |
| FDCB d 50 | BIT 2,(IY+d)* | FDCB d 9A | RES 3,(IY+d),D* | FDCB d E4 | SET 4,(IY+d),H* |
| FDCB d 51 | BIT 2,(IY+d)* | FDCB d 9B | RES 3,(IY+d),E* | FDCB d E5 | SET 4,(IY+d),L* |
| FDCB d 52 | BIT 2,(IY+d)* | FDCB d 9C | RES 3,(IY+d),H* | FDCB d E6 | SET 4,(IY+d) |
| FDCB d 53 | BIT 2,(IY+d)* | FDCB d 9D | RES 3,(IY+d),L* | FDCB d E7 | SET 4,(IY+d),A* |
| FDCB d 54 | BIT 2,(IY+d)* | FDCB d 9E | RES 3,(IY+d) | FDCB d E8 | SET 5,(IY+d),B* |
| FDCB d 55 | BIT 2,(IY+d)* | FDCB d 9F | RES 3,(IY+d),A* | FDCB d E9 | SET 5,(IY+d),C* |
| FDCB d 56 | BIT 2,(IY+d) | FDCB d A0 | RES 4,(IY+d),B* | FDCB d EA | SET 5,(IY+d),D* |
| FDCB d 57 | BIT 2,(IY+d)* | FDCB d A1 | RES 4,(IY+d),C* | FDCB d EB | SET 5,(IY+d),E* |
| FDCB d 58 | BIT 3,(IY+d)* | FDCB d A2 | RES 4,(IY+d),D* | FDCB d EC | SET 5,(IY+d),H* |
| FDCB d 59 | BIT 3,(IY+d)* | FDCB d A3 | RES 4,(IY+d),E* | FDCB d ED | SET 5,(IY+d),L* |
| FDCB d 5A | BIT 3,(IY+d)* | FDCB d A4 | RES 4,(IY+d),H* | FDCB d EE | SET 5,(IY+d) |
| FDCB d 5B | BIT 3,(IY+d)* | FDCB d A5 | RES 4,(IY+d),L* | FDCB d EF | SET 5,(IY+d),A* |

CHAPTER 9. INSTRUCTIONS SORTED BY OPCODE

| | | | | | |
|-----------|-------------------|-----------|-------------------|------|-------------|
| FDCB d F0 | SET 6, (IY+d), B* | FDCB d F8 | SET 7, (IY+d), B* | FDE1 | POP IY |
| FDCB d F1 | SET 6, (IY+d), C* | FDCB d F9 | SET 7, (IY+d), C* | FDE3 | EX (SP), IY |
| FDCB d F2 | SET 6, (IY+d), D* | FDCB d FA | SET 7, (IY+d), D* | FDE5 | PUSH IY |
| FDCB d F3 | SET 6, (IY+d), E* | FDCB d FB | SET 7, (IY+d), E* | FDE9 | JP (IY) |
| FDCB d F4 | SET 6, (IY+d), H* | FDCB d FC | SET 7, (IY+d), H* | FDF9 | LD SP, IY |
| FDCB d F5 | SET 6, (IY+d), L* | FDCB d FD | SET 7, (IY+d), L* | FE n | CP n |
| FDCB d F6 | SET 6, (IY+d) | FDCB d FE | SET 7, (IY+d) | FF | RST 38H |
| FDCB d F7 | SET 6, (IY+d), A* | FDCB d FF | SET 7, (IY+d), A* | | |

Chapter 10

GNU Free Documentation License

Version 1.1, March 2000

Copyright © 2000 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other written document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

10.1 Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this

CHAPTER 10. GNU FREE DOCUMENTATION LICENSE

License. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, L^AT_EX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

10.2 Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this

License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

10.3 Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

10.4 Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You

CHAPTER 10. GNU FREE DOCUMENTATION LICENSE

may use the same title as a previous version if the original publisher of that version gives permission.

- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.
- Include an unaltered copy of this License.
- Preserve the section entitled “History”, and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- In any section entitled “Acknowledgements” or “Dedications”, preserve the section’s title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- Delete any section entitled “Endorsements”. Such a section may not be included in the Modified Version.
- Do not retitle any existing section as “Endorsements” or to conflict in title with any Invariant Section.

CHAPTER 10. GNU FREE DOCUMENTATION LICENSE

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

10.5 Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

10.6 Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

10.7 Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an “aggregate”, and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

10.8 Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

10.9 Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10.10 Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

CHAPTER 10. GNU FREE DOCUMENTATION LICENSE

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.