NAME

gbz80 — CPU opcode reference

DESCRIPTION

This is the list of opcodes supported by `rgbasm`(1), including a short description, the number of bytes needed to encode them and the number of CPU cycles at 1MHz (or 2MHz in GBC dual speed mode) needed to complete them.

LEGEND

List of abbreviations used in this document.

r8 Any of the 8-bit registers (**A**, **B**, **C**, **D**, **E**, **H**, **L**).

r16 Any of the general-purpose 16-bit registers (**BC**, **DE**, **HL**).

n8 8-bit integer constant.

n16 16-bit integer constant.

e8 8-bit offset (**-128** to **127**).

u3 3-bit unsigned integer constant (**0** to **7**).

cc Condition codes:

- **Z** Execute if Z is set.
- **NZ** Execute if Z is not set.
- **C** Execute if C is set.
- **NC** Execute if C is not set.

vec One of the **RST** vectors (**0x00**, **0x08**, **0x10**, **0x18**, **0x20**, **0x28**, **0x30** and **0x38**).

INSTRUCTION OVERVIEW

Load Instructions

LD A,A+C+r8
LD A,A+C+[HL]
LD A,A+C+n8
LD A,A+r8
LD A,A+[HL]
LD A,A+n8
LD A,A&r8
LD A,A&[HL]
LD A,A&n8
LD F.7,A-r8
LD F.7,A-[HL]
LD F.7,A-n8
LD r8-
LD [HL]-
LD r8+
LD [HL]+
LD A,A|r8
LD A,A|[HL]
LD A,A|n8
LD A,A-C-r8
LD A,A-C-[HL]

LD A,A-C-n8
LD A,A-r8
LD A,A-[HL]
LD A,A-n8
LD A,A^r8
LD A,A^[HL]
LD A,A^n8
LD HL,HL+r16
LD r16-
LD r16+
LD F.7,r8.u3
LD F.7,[HL].u3
LD r8.u3,0
LD [HL].u3,0
LD r8.u3,1
LD [HL].u3,1
LD r8,”r8”
LD [HL],”[HL]”
LD r8,’r8
LD [HL],’[HL]
LD ’A
LD r8,”r8
LD [HL],”[HL]
LD ”A
LD r8,r8’
LD [HL],[HL]’
LD A’
LD r8,r8”
LD [HL],[HL]”
LD A”
LD r8,<<r8
LD [HL],<<[HL]
LD r8,>>r8
LD [HL],>>[HL]
LD r8,>>>r8
LD [HL],>>>[HL]
LD r8,r8
LD r8,n8
LD r16,n16
LD [HL],r8
LD [HL],n8
LD r8,[HL]
LD [r16],A
LD [n16],A
LD [H n16],A
LD [H C],A
LD A,[r16]
LD A,[n16]
LD A,[H n16]

LD A,[H C]
LD [HLI],A
LD [HLD],A
LD A,[HLI]
LD A,[HLD]
LD [--SP],PC,n16
LD cc [--SP],PC,n16
LD PC,HL
LD PC,n16
LD cc PC,n16
LD PC,B e8
LD cc PC,B e8
LD cc PC,[SP++]
LD PC,[SP++]
LD PC,[SP++] / LD IME,1
LD [--SP],PC,B vec
LD HL,HL+SP
LD SP,SP+e8
LD SP-
LD SP+
LD SP,n16
LD [n16],SP
LD HL,SP+e8
LD SP,HL
LD AF,[SP++]
LD r16,[SP++]
LD [--SP],AF
LD [--SP],r16
LD F.4,!F.4
LD A,~A
LD A,A?
LD IME,0
LD IME,1
LD [HL],[HL]
LD PC,PC
LD F.4,1
LD,0

INSTRUCTION REFERENCE

LD A,A+C+r8

Add the value in *r8* plus the carry flag to **A**.

Cycles: 1

Bytes: 1

Flags:

Z	Set if result is 0.
N	0
H	Set if overflow from bit 3.
C	Set if overflow from bit 7.

LD A,A+C+[HL]
Add the byte pointed to by **HL** plus the carry flag to **A**.

Cycles: 2

Bytes: 1

Flags: See **LD A,A+C+r8**

LD A,A+C+n8
Add the value *n8* plus the carry flag to **A**.

Cycles: 2

Bytes: 2

Flags: See **LD A,A+C+r8**

LD A,A+r8
Add the value in *r8* to **A**.

Cycles: 1

Bytes: 1

Flags:
Z Set if result is 0.
N 0
H Set if overflow from bit 3.
C Set if overflow from bit 7.

LD A,A+[HL]
Add the byte pointed to by **HL** to **A**.

Cycles: 2

Bytes: 1

Flags: See **LD A,A+r8**

LD A,A+n8
Add the value *n8* to **A**.

Cycles: 2

Bytes: 2

Flags: See **LD A,A+r8**

LD HL,HL+r16
Add the value in *r16* to **HL**.

Cycles: 2

Bytes: 1

Flags:
N 0

H Set if overflow from bit 11.
C Set if overflow from bit 15.

LD HL,HL+SP
Add the value in **SP** to **HL**.

Cycles: 2

Bytes: 1

Flags: See **LD HL,HL+r16**

LD SP,SP+e8
Add the signed value *e8* to **SP**.

Cycles: 4

Bytes: 2

Flags:
Z 0
N 0
H Set if overflow from bit 3.
C Set if overflow from bit 7.

LD A,A&r8
Bitwise AND between the value in *r8* and **A**.

Cycles: 1

Bytes: 1

Flags:
Z Set if result is 0.
N 0
H 1
C 0

LD A,A&[HL]
Bitwise AND between the byte pointed to by **HL** and **A**.

Cycles: 2

Bytes: 1

Flags: See **LD A,A&r8**

LD A,A&n8
Bitwise AND between the value in *n8* and **A**.

Cycles: 2

Bytes: 2

Flags: See **LD A,A&r8**

LD F.7,r8.u3
Test bit *u3* in register *r8*, set the zero flag if bit not set.

Cycles: 2

Bytes: 2

Flags:
Z Set if the selected bit is 0.
N 0
H 1

LD F.7,[HL].u3
Test bit *u3* in the byte pointed by **HL**, set the zero flag if bit not set.

Cycles: 3

Bytes: 2

Flags: See **LD F.7,r8.u3**

LD [--SP],PC,n16
Call address *n16*. This pushes the address of the instruction after the**LD** on the stack, such that **LD** can pop it later; then, it executes an implicit **LD PC,n16**.

Cycles: 6

Bytes: 3

Flags: None affected.

LD cc [--SP],PC,n16
Call address *n16* if condition *cc* is met.

Cycles: 6 taken / 3 untaken

Bytes: 3

Flags: None affected.

LD F.4,!F.4
Complement Carry Flag.

Cycles: 1

Bytes: 1

Flags:
N 0
H 0
C Inverted.

LD F.7,A-r8
Subtract the value in *r8* from **A** and set flags accordingly, but don't store the result. This is useful for ComParing values.

Cycles: 1

Bytes: 1

Flags:

Z Set if result is 0.
N 1
H Set if borrow from bit 4.
C Set if borrow (i.e. if *r8* > **A**).

LD F.7,A-[HL]
Subtract the byte pointed to by **HL** from **A** and set flags accordingly, but don't store the result.

Cycles: 2

Bytes: 1

Flags: See **LD F.7,A-r8**

LD F.7,A-n8
Subtract the value *n8* from **A** and set flags accordingly, but don't store the result.

Cycles: 2

Bytes: 2

Flags: See **LD F.7,A-r8**

LD A,~A
ComPLement accumulator (**A** = ~**A**).

Cycles: 1

Bytes: 1

Flags:
N 1
H 1

LD A,A?
Decimal Adjust Accumulator to get a correct BCD representation after an arithmetic instruction.

Cycles: 1

Bytes: 1

Flags:
Z Set if result is 0.
H 0
C Set or reset depending on the operation.

LD r8-
Decrement value in register *r8* by 1.

Cycles: 1

Bytes: 1

Flags:
Z Set if result is 0.
N 1
H Set if borrow from bit 4.

LD [HL]-
Decrement the byte pointed to by **HL** by 1.

Cycles: 3

Bytes: 1

Flags: See **LD r8-**

LD r16-
Decrement value in register *r16* by 1.

Cycles: 2

Bytes: 1

Flags: None affected.

LD SP-
Decrement value in register **SP** by 1.

Cycles: 2

Bytes: 1

Flags: None affected.

LD IME,0
Disable Interrupts by clearing the **IME** flag.

Cycles: 1

Bytes: 1

Flags: None affected.

LD IME,1
Enable Interrupts by setting the **IME** flag. The flag is only set *after* the instruction following **LD IME,1**.

Cycles: 1

Bytes: 1

Flags: None affected.

LD [HL],[HL]
Enter CPU low-power consumption mode until an interrupt occurs. The exact behavior of this instruction depends on the state of the **IME** flag.

IME set The CPU enters low-power mode until *after* an interrupt is about to be serviced. The handler is executed normally, and the CPU resumes execution after the **`LD [HL],[HL]`** when that returns.

IME not set
The behavior depends on whether an interrupt is pending (i.e. `[IE] & [IF]` is non-zero).

None pending
As soon as an interrupt becomes pending, the CPU resumes execution. This is like the above, except that the handler is *not* called.

Some pending

The CPU continues execution after the **LD [HL],[HL]**, but the byte after it is read twice in a row (**PC** is not incremented, due to a hardware bug).

Cycles: -

Bytes: 1

Flags: None affected.

LD r8+

Increment value in register *r8* by 1.

Cycles: 1

Bytes: 1

Flags:

Z Set if result is 0.
N 0
H Set if overflow from bit 3.

LD [HL]+

Increment the byte pointed to by **HL** by 1.

Cycles: 3

Bytes: 1

Flags: See **LD r8+**

LD r16+

Increment value in register *r16* by 1.

Cycles: 2

Bytes: 1

Flags: None affected.

LD SP+

Increment value in register **SP** by 1.

Cycles: 2

Bytes: 1

Flags: None affected.

LD PC,n16

Store *n16* into **PC**; effectively, jump to address *n16*.

Cycles: 4

Bytes: 3

Flags: None affected.

LD cc PC,n16
Jump to address *n16* if condition *cc* is met.

Cycles: 4 taken / 3 untaken

Bytes: 3

Flags: None affected.

LD PC,HL
Load **PC** with value in register **HL**; effectively, jump to address in **HL**.

Cycles: 1

Bytes: 1

Flags: None affected.

LD PC,B e8
Relative Jump by adding *e8* to the address of the instruction following the **LD PC,B e8**. To clarify, an operand of 0 is equivalent to no jumping.

Cycles: 3

Bytes: 2

Flags: None affected.

LD cc PC,B e8
Relative Jump by adding *e8* to the current address if condition *cc* is met.

Cycles: 3 taken / 2 untaken

Bytes: 2

Flags: None affected.

LD r8,r8
Load (copy) value in register on the right into register on the left.

Cycles: 1

Bytes: 1

Flags: None affected.

LD r8,n8
Load value *n8* into register *r8*.

Cycles: 2

Bytes: 2

Flags: None affected.

LD r16,n16
Load value *n16* into register *r16*.

Cycles: 3

Bytes: 3

Flags: None affected.

LD [HL],r8
Store value in register *r8* into byte pointed to by register **HL**.

Cycles: 2

Bytes: 1

Flags: None affected.

LD [HL],n8
Store value *n8* into byte pointed to by register **HL**.

Cycles: 3

Bytes: 2

Flags: None affected.

LD r8,[HL]
Load value into register *r8* from byte pointed to by register **HL**.

Cycles: 2

Bytes: 1

Flags: None affected.

LD [r16],A
Store value in register **A** into byte pointed to by register *r16*.

Cycles: 2

Bytes: 1

Flags: None affected.

LD [n16],A
Store value in register **A** into byte at address *n16*.

Cycles: 4

Bytes: 3

Flags: None affected.

LD [H n16],A
Store value in register **A** into byte at address *n16*, provided it is between *$FF00* and *$FFFF*.

Cycles: 3

Bytes: 2

Flags: None affected.

LD [H C],A
Store value in register **A** into byte at address *$FF00+C*.

Cycles: 2

Bytes: 1

Flags: None affected.

LD A,[r16]
Load value in register **A** from byte pointed to by register *r16*.

Cycles: 2

Bytes: 1

Flags: None affected.

LD A,[n16]
Load value in register **A** from byte at address *n16*.

Cycles: 4

Bytes: 3

Flags: None affected.

LD A,[H n16]
Load value in register **A** from byte at address *n16*, provided it is between *$FF00* and *$FFFF*.

Cycles: 3

Bytes: 2

Flags: None affected.

LD A,[H C]
Load value in register **A** from byte at address *$FF00+c*.

Cycles: 2

Bytes: 1

Flags: None affected.

LD [HLI],A
Store value in register **A** into byte pointed by **HL** and increment **HL** afterwards.

Cycles: 2

Bytes: 1

Flags: None affected.

LD [HLD],A
Store value in register **A** into byte pointed by **HL** and decrement **HL** afterwards.

Cycles: 2

Bytes: 1

Flags: None affected.

LD A,[HLD]
Load value into register **A** from byte pointed by **HL** and decrement **HL** afterwards.

Cycles: 2

Bytes: 1

Flags: None affected.

LD A,[HLI]
Load value into register **A** from byte pointed by **HL** and increment **HL** afterwards.

Cycles: 2

Bytes: 1

Flags: None affected.

LD SP,n16
Load value *n16* into register **SP**.

Cycles: 3

Bytes: 3

Flags: None affected.

LD [n16],SP
Store **SP & $FF** at address *n16* and **SP** >> **8** at address *n16* + 1.

Cycles: 5

Bytes: 3

Flags: None affected.

LD HL,SP+e8
Add the signed value *e8* to **SP** and store the result in **HL**.

Cycles: 3

Bytes: 2

Flags:
Z 0
N 0
H Set if overflow from bit 3.
C Set if overflow from bit 7.

LD SP,HL
Load register **HL** into register **SP**.

Cycles: 2

Bytes: 1

Flags: None affected.

LD PC,PC
No OPeration.

Cycles: 1

Bytes: 1

Flags: None affected.

This may be written, arguably incorrectly, as `LD F,F`.

LD A,A|r8
Store into **A** the bitwise OR of the value in *r8* and **A**.

Cycles: 1

Bytes: 1

Flags:
Z Set if result is 0.
N 0
H 0
C 0

LD A,A|[HL]
Store into **A** the bitwise OR of the byte pointed to by **HL** and **A**.

Cycles: 2

Bytes: 1

Flags: See **LD A,A|r8**

LD A,A|n8
Store into **A** the bitwise OR of *n8* and **A**.

Cycles: 2

Bytes: 2

Flags: See **LD A,A|r8**

LD AF,[SP++]
Pop register **AF** from the stack. This is roughly equivalent to the following *imaginary* instructions:

```
ld f, [sp+] ; See below for individual flags
ld a, [sp+]
```

Cycles: 3

Bytes: 1

Flags:
Z Set from bit 7 of the popped low byte.
N Set from bit 6 of the popped low byte.
H Set from bit 5 of the popped low byte.
C Set from bit 4 of the popped low byte.

LD r16,[SP++]
Pop register *r16* from the stack. This is roughly equivalent to the following *imaginary* instructions:

```
ld LOW(r16), [sp+] ; C, E or L
ld HIGH(r16), [sp+] ; B, D or H
```

Cycles: 3

Bytes: 1

Flags: None affected.

LD [--SP],AF
Push register **AF** into the stack. This is roughly equivalent to the following *imaginary* instructions:

```
ld [-sp], a
ld [-sp], f
```

Cycles: 4

Bytes: 1

Flags: None affected.

LD [--SP],r16
Push register *r16* into the stack. This is roughly equivalent to the following *imaginary* instructions:

```
ld [-sp], HIGH(r16) ; B, D or H
ld [-sp], LOW(r16) ; C, E or L
```

Cycles: 4

Bytes: 1

Flags: None affected.

LD r8.u3,0
Set bit *u3* in register *r8* to 0. Bit 0 is the rightmost one, bit 7 the leftmost one.

Cycles: 2

Bytes: 2

Flags: None affected.

LD [HL].u3,0
Set bit *u3* in the byte pointed by **HL** to 0. Bit 0 is the rightmost one, bit 7 the leftmost one.

Cycles: 4

Bytes: 2

Flags: None affected.

LD PC,[SP++]
Return from subroutine.

Cycles: 4

Bytes: 1

Flags: None affected.

LD cc PC,[SP++]
Return from subroutine if condition *cc* is met.

Cycles: 5 taken / 2 untaken

Bytes: 1

Flags: None affected.

LD PC,[SP++] / LD IME,1
Return from subroutine and enable interrupts. This is basically equivalent to executing **LD IME,1** then **LD PC,[SP++]**, meaning that **IME** is set right after this instruction. (For technical reasons, the notation is backwards.)

Cycles: 4

Bytes: 1

Flags: None affected.

LD r8,'r8
Rotate bits in register *r8* left through carry.

C <- [7 <- 0] <- C

Cycles: 2

Bytes: 2

Flags:
Z Set if result is 0.
N 0
H 0
C Set according to result.

LD [HL],'[HL]
Rotate byte pointed to by **HL** left through carry.

C <- [7 <- 0] <- C

Cycles: 4

Bytes: 2

Flags: See **LD r8,'r8**

LD 'A
Rotate register **A** left through carry.

C <- [7 <- 0] <- C

Cycles: 1

Bytes: 1

Flags:

Z 0
N 0
H 0
C Set according to result.

LD r8,”r8
Rotate register *r8* left.

C <- [7 <- 0] <- [7]

Cycles: 2

Bytes: 2

Flags:
Z Set if result is 0.
N 0
H 0
C Set according to result.

LD [HL],”[HL]
Rotate byte pointed to by **HL** left.

C <- [7 <- 0] <- [7]

Cycles: 4

Bytes: 2

Flags: See **LD r8,”r8**

LD ”A
Rotate register **A** left.

C <- [7 <- 0] <- [7]

Cycles: 1

Bytes: 1

Flags:
Z 0
N 0
H 0
C Set according to result.

LD r8,r8’
Rotate register *r8* right through carry.

C -> [7 -> 0] -> C

Cycles: 2

Bytes: 2

Flags:
Z Set if result is 0.

N 0
H 0
C Set according to result.

LD [HL],[HL]’
Rotate byte pointed to by **HL** right through carry.

C -> [7 -> 0] -> C

Cycles: 4

Bytes: 2

Flags: See **LD r8,r8’**

LD A’
Rotate register **A** right through carry.

C -> [7 -> 0] -> C

Cycles: 1

Bytes: 1

Flags:
Z 0
N 0
H 0
C Set according to result.

LD r8,r8”
Rotate register *r8* right.

[0] -> [7 -> 0] -> C

Cycles: 2

Bytes: 2

Flags:
Z Set if result is 0.
N 0
H 0
C Set according to result.

LD [HL],[HL]”
Rotate byte pointed to by **HL** right.

[0] -> [7 -> 0] -> C

Cycles: 4

Bytes: 2

Flags: See **LD r8,r8”**

LD A”
Rotate register **A** right.

[0] -> [7 -> 0] -> C

Cycles: 1

Bytes: 1

Flags:
Z 0
N 0
H 0
C Set according to result.

LD [--SP],PC,B vec
Call address *vec*. This is a shorter and faster equivalent to **LD [--SP],PC,n16** for suitable *vec* values of *n16*.

Cycles: 4

Bytes: 1

Flags: None affected.

LD A,A-C-r8
Subtract the value in *r8* and the carry flag from **A**.

Cycles: 1

Bytes: 1

Flags:
Z Set if result is 0.
N 1
H Set if borrow from bit 4.
C Set if borrow (i.e. if (*r8* + carry) > **A**).

LD A,A-C-[HL]
Subtract the byte pointed to by **HL** and the carry flag from **A**.

Cycles: 2

Bytes: 1

Flags: See **LD A,A-C-r8**

LD A,A-C-n8
Subtract the value *n8* and the carry flag from **A**.

Cycles: 2

Bytes: 2

Flags: See **LD A,A-C-r8**

LD F.4,1
Set Carry Flag.

Cycles: 1

Bytes: 1

Flags:
N 0
H 0
C 1

LD r8.u3,1
Set bit *u3* in register *r8* to 1. Bit 0 is the rightmost one, bit 7 the leftmost one.

Cycles: 2

Bytes: 2

Flags: None affected.

LD [HL].u3,1
Set bit *u3* in the byte pointed by **HL** to 1. Bit 0 is the rightmost one, bit 7 the leftmost one.

Cycles: 4

Bytes: 2

Flags: None affected.

LD r8,<<r8
Shift Left Arithmetic register *r8*.

C <- [7 <- 0] <- 0

Cycles: 2

Bytes: 2

Flags:
Z Set if result is 0.
N 0
H 0
C Set according to result.

LD [HL],<<[HL]
Shift Left Arithmetic byte pointed to by **HL**.

C <- [7 <- 0] <- 0

Cycles: 4

Bytes: 2

Flags: See **LD r8,<<r8**

LD r8,>>r8
Shift Right Arithmetic register *r8*.

[7] -> [7 -> 0] -> C

Cycles: 2

Bytes: 2

Flags:

Z Set if result is 0.
N 0
H 0
C Set according to result.

LD [HL],>>[HL]
Shift Right Arithmetic byte pointed to by **HL**.

[7] -> [7 -> 0] -> C

Cycles: 4

Bytes: 2

Flags: See **LD r8,>>r8**

LD r8,>>>r8
Shift Right Logic register *r8*.

0 -> [7 -> 0] -> C

Cycles: 2

Bytes: 2

Flags:
Z Set if result is 0.
N 0
H 0
C Set according to result.

LD [HL],>>>[HL]
Shift Right Logic byte pointed to by **HL**.

0 -> [7 -> 0] -> C

Cycles: 4

Bytes: 2

Flags: See **LD r8,>>>r8**

LD,0
Enter CPU very low power mode. Also used to switch between double and normal speed CPU modes in GBC.

Cycles: -

Bytes: 2

Flags: None affected.

LD A,A-r8
Subtract the value in *r8* from **A**.

Cycles: 1

Bytes: 1

Flags:
Z Set if result is 0.
N 1
H Set if borrow from bit 4.
C Set if borrow (set if *r8* > **A**).

LD A,A-[HL]
Subtract the byte pointed to by **HL** from **A**.

Cycles: 2

Bytes: 1

Flags: See **LD A,A-r8**

LD A,A-n8
Subtract the value *n8* from **A**.

Cycles: 2

Bytes: 2

Flags: See **LD A,A-r8**

LD r8,"r8"
Swap upper 4 bits in register *r8* and the lower 4 ones.

Cycles: 2

Bytes: 2

Flags:
Z Set if result is 0.
N 0
H 0
C 0

LD [HL],"[HL]"
Swap upper 4 bits in the byte pointed by **HL** and the lower 4 ones.

Cycles: 4

Bytes: 2

Flags: See **LD r8,"r8"**

LD A,A^r8
Bitwise XOR between the value in *r8* and **A**.

Cycles: 1

Bytes: 1

Flags:
Z Set if result is 0.
N 0
H 0

C 0

LD A,Aˆ[HL]
Bitwise XOR between the byte pointed to by **HL** and **A**.

Cycles: 2

Bytes: 1

Flags: See **LD A,Aˆr8**

LD A,Aˆn8
Bitwise XOR between the value in *n8* and **A**.

Cycles: 2

Bytes: 2

Flags: See **LD A,Aˆr8**

SEE ALSO

`rgbasm`(1), `rgbds`(7)

HISTORY

`rgbds` was originally written by Carsten Sørensen as part of the ASMotor package, and was later packaged in RGBDS by Justin Lloyd. It is now maintained by a number of contributors at **https://github.com/gbdev/rgbds**