eZ80

Fourth-Generation Z80 Processor Core

Processor Description

PS002200-ZMP0999

ZiLOG Worldwide Headquarters • 910 E. Hamilton Avenue • Campbell, CA 95008
Telephone: 408.558.8500 • Fax: 408.558.8300 • Internet: http://www.ZiLOG.com

©1999 by ZiLOG, Inc. All rights reserved. Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZiLOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZiLOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. Except with the express written approval of ZiLOG, use of information, devices, or technology as critical components of life support systems is not authorized. No licenses are conveyed, implicitly or otherwise, by this document under any intellectual property rights.

GENERAL DESCRIPTION

The eZ80™ is ZiLOG's next-generation Z80™ processor. The eZ80 provides 16 times the performance of a traditional Z80. The multiple operating modes of the processor allows Z80 and Z180 code to be run without change in the same application with new code, that takes advantage of the eZ80's 16-MB linear addressing space and enhanced instruction set. These features provide customers performance comparable to 16-bit processors with the form factor and power savings of an 8-bit processor. At the same time, the eZ80 remains 100% Z80 code-compatible, reducing customer development time.

The eZ80 also features a Multiply and Accumulate engine, which enables customers to attack signal-processing applications that require polynomial calculations, such as basic filters.

The eZ80 is internet-ready. ZiLOG can provide a complete TCP/IP stack, allowing for rapid internet connectivity.

The eZ80 also features ZiLOG's Debug Interface (ZDI). This two-pin interface allows advanced debugging features without the cost and difficulty and uncertainty of an in-circuit emulator.

The eZ80 is a licensable soft core, allowing rapid integration into designs.

DETAILED DESCRIPTION

Z80 High-Performance Microprocessor Core. The eZ80 is one of the fastest 8-bit CPUs available today, executing code 4 times faster than a standard Z80 operating at the same clock speed. The increased processing efficiency can be used to improve available bandwidth or to decrease power consumption.

Both the increased clock speed and processor efficiency features provides eZ80 customers 16 times the processing performance. This processing power rivals performance customers would normally expect from 16-bit microprocessors.

16 MB Linear Address. The eZ80 is also the first 8-bit microprocessor to support 16 MB linear addressing—a feature that addresses large memories that support complex software applications.

Each software module, or each task under a real-time executive or operating system, can operate in Z80-compatible (64 KB) mode, Z80180-compatible mode (1 MB MMU) mode, or full 24-bit (16 MB) address mode.

Internet-Ready. A complete TCP/IP stack is also offered so customers can design products that connect to the Internet.

Multiply and Accumulate. A Multiply and Accumulate engine operates in parallel with the eZ80 processor to calculate a sum of products that is the core of digital signal processing. The MAC provides 16x16 multiply and 40-bit accumulation.

ZDI. The ZiLOG Debug Interface is a 2-pin communication port. When used with the ZiLOG Develop Suite (ZDS) software, ZDI provides on-chip emulation.

ARCHITECTURAL OVERVIEW

The eZ80 is ZiLOG's fourth-generation Z80 processor core. It is the basis of a new family of integrated microprocessors, and includes the following features:

- Upward-code-compatible from Z80 & Z180
- Several address-generation modes including 24-bit linear addressing
- 24-bit registers and ALU
- One-clock-minimum bus cycles
- Optional autonomous Multiply-Accumulate engine for DSP applications

BLOCK DIAGRAM

Figure 1 is a block diagram of the eZ80.

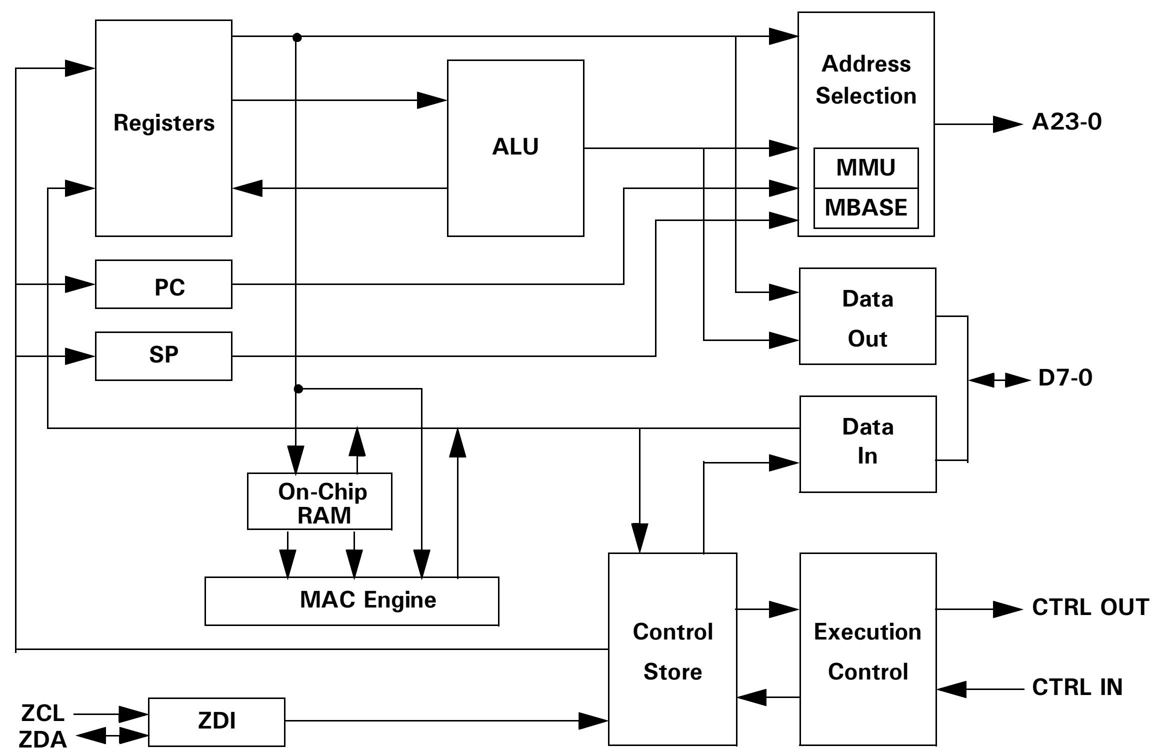

FIGURE 1. EZ80 BLOCK DIAGRAM

PIN DESCRIPTIONS

Figure 2 illustrates the logic diagram of the eZ80. Table 1 describes the processor and device pins.

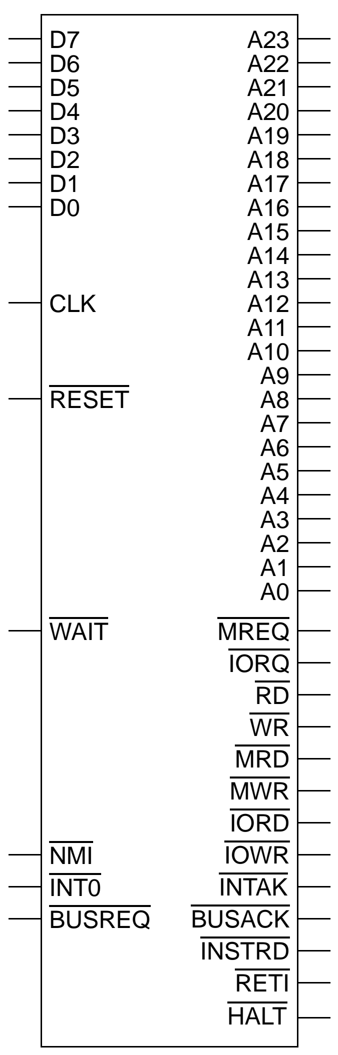

FIGURE 2. eZ80 LOGIC DIAGRAM

TABLE 1. PROCESSOR AND DEVICE PIN DESCRIPTIONS

Symbol	Function	Type	Description
A23–0	Address Bus	Bidirectional, 3-state	These lines select a location in memory or I/O space to be read or written. The eZ80 does not drive these lines during Reset or external bus acknowledge cycles.
BUSACK	Bus Acknowledge	Output, active Low	The eZ80 responds to a Low on BUSREQ, by 3-stating the address, data, and control signals and driving this line Low.
BUSREQ	Bus request	Input, active Low	External devices can force the eZ80 to release the bus for their use, by driving this line Low.
CLK	Clock	Input	The master clock of the eZ80.
D7–0	Data Bus	Bidirectional, 3-state	These lines transfer information to and from I/O and memory devices. The eZ80 drives these lines only during write cycles.
HALT	Halt	Output, active Low	A low on this pin indicates that the eZ80 is stopped because of a HALT or SLP instruction.
INSTRD	Instruction Read	Output, active Low, 3-state	INSTRD Low (with MREQ and MRD Low) indicates that the eZ80 is fetching an instruction from memory. The eZ80 does not drive this line during Reset, nor during bus acknowledge cycles.
INT0	Interrupt Request 0	Input, active Low	External devices can drive this line Low to request an interrupt. The processor responds to this request at the end of the current instruction cycle if it is enabled, and the NMI and BUSREQ signals are inactive.
INTAK	Interrupt Acknowledge	Output, active Low, 3-state	INTAK Low indicates that the eZ80 is acknowledging an interrupt request on INT0. The eZ80 does not drive this line during Reset, nor during bus acknowledge cycles.
IORD	I/O Read	Output, active Low, 3-state	IORD Low indicates that the eZ80 is reading data from a location in I/O space. The addressed I/O device uses this signal to gate data onto the data bus. The eZ80 does not drive this line during Reset, nor during bus acknowledge cycles.
IORQ	I/O Request	Output, active Low, 3-state	IORQ Low indicates that the eZ80 is accessing a location in I/O space. The RD and WR pins indicate the type of access. The eZ80 does not drive this line during Reset, nor during bus acknowledge cycles.
IOWR	I/O Write	Output, active Low, 3-state	IOWR Low indicates that D7–0 hold data to be stored at the addressed I/O location. The eZ80 does not drive this line during Reset, nor during bus acknowledge cycles.

TABLE 1. PROCESSOR AND DEVICE PIN DESCRIPTIONS (CONTINUED)

Symbol	Function	Type	Description
$\overline{\text{MRD}}$	Memory Read	Output, active Low, 3-state	$\overline{\text{MRD}}$ Low indicates that the eZ80 is reading data from a location in memory space. The addressed memory uses this signal to gate data onto the data bus. The eZ80 does not drive this line during Reset, nor during bus acknowledge cycles.
$\overline{\text{MREQ}}$	Memory Request	Output, active Low, 3-state	$\overline{\text{MREQ}}$ Low indicates that the eZ80 is accessing a location in memory. The $\overline{\text{RD}}$, $\overline{\text{WR}}$, and $\overline{\text{INSTRD}}$ pins indicate the type of access. The eZ80 does not drive this line during Reset, nor during bus acknowledge cycles.
$\overline{\text{MWR}}$	Memory Write	Output, active Low, 3-state	$\overline{\text{MWR}}$ low indicates that D7–0 hold data to be stored at the addressed memory location. The eZ80 does not drive this line during Reset, nor during bus acknowledge cycles.
$\overline{\text{NMI}}$	Nonmaskable Interrupt	Input, falling-edge active	$\overline{\text{NMI}}$ has a higher priority than $\overline{\text{INT0}}$ and is always recognized at the end of an instruction, regardless of the state of the interrupt enable flip-flops. This signal forces processor execution to location 0066H. This input includes a Schmitt trigger to allow RC rise times.
$\overline{\text{RESET}}$	Master Reset	Input/Output, active Low	This signal is used to initialize the eZ80 and other devices in the system. This input must be held Low until the clock is stable. This input includes a Schmitt trigger, allowing RC rise times.
$\overline{\text{RETI}}$	Return from Interrupt	Output, active Low	A Low on this line indicates that the eZ80 is executing an RETI instruction.
VDD	Power Supply		These pins carry power to the device. They must be tied to the same voltage externally.
VSS	Ground		These pins are the ground references for the device. They must be tied to the same voltage externally.
$\overline{\text{WAIT}}$	Wait	Input, active Low	External devices can extend bus cycles to more than one clock, by driving this line low.

OPERATIONAL DESCRIPTION

This section describes, using text, tables, and figures, how the various parts of the eZ80 operate. This description is presented from the processor outward to the peripherals. Refer to the corresponding section of “I/O Registers” on page 20, which describes the eZ80’s I/O registers.

PROCESSOR DESCRIPTION

The eZ80 is an 8-bit microprocessor that performs certain 16- or 24-bit operations. In both data sizes, the processor includes an accumulator. Register A is the accumulator for 8-bit operations, and the HL register pair is the accumulator for 16- and 24-bit operations.

Processor Program Registers

In addition to register A, there are six more 8-bit registers named B, C, D, E, H, and L, which can also be operated as register pairs BC, DE, and HL. Flag register F completes the basic register bank.

Two of these basic register banks are included in all Z80 and Z180 processors. High-speed exchange between these banks can be used by a program internally, or one bank can be allocated to the mainline program and the other to interrupt service routines.

Two Index registers IX and IY allow *base and displacement* addressing in memory. IX and IY are not included in the register banks on the eZ80. They are independent of the register banks.

The eZ80 expands the width of the BC, DE, HL, IX, and IY registers from 16 to 24 bits. The Arithmetic/Logic Unit and internal data paths are similarly expanded to 24 bits.

Processor Control Registers

In addition to the data-oriented registers described above, the eZ80 processor includes several other control registers. Unlike the registers in I/O space that are described in "I/O Registers" on page 20, these control registers have no addresses, but are used implicitly in certain processor operations.

Program Counter (PC). This 16- or 24-bit register tracks program execution by the processor, which automatically increments PC while fetching instructions. The processor stores PC on the stack when it executes a CALL or RST instruction, or an interrupt or Trap occurs. It loads PC with a new value when it executes a JUMP, CALL, RST, or RET instruction, and when an interrupt, Trap, or Reset occurs. PC resets to 0000.

Stack Pointer (SPS or SPL). SPS is a 16-bit register that is used when the ADL bit is cleared, while SPL is a 24-bit register that is used when ADL is set. The processor decrements the current SP register by 2 or 3, and stores a 16- or 24-bit value in memory at this updated address, when it executes a PUSH, CALL, or RST instruction, and when an interrupt or Trap occurs. The processor fetches a 16- or 24-bit value from memory at the address in SP, and then increments SP by 2 or 3, when it executes a POP, RET, RETI, or RETN instruction. Software can store the SP value in memory, load SP from memory or another register, or load it with a constant/immediate value. Further, software can add or subtract the value in SP to or from another register, and can increment or decrement SP. Finally, software can exchange the 16- or 24-bit value in memory, to which SP currently points, with the contents of a 16- or 24-bit register. SP resets to 0000.

Flags (F). The processor includes two Flag registers each containing six bits, named Zero (Z), Carry (CF), Sign (S), Parity or Overflow (P/V), Half-Carry (HC) and Add/Subtract (N). Certain flags are automatically updated as part of executing certain instructions. Subsequent instructions can then use the flags, either as an operand (ADC, SBC, DAA), or to determine whether to perform a JUMP, CALL, or RET operation. The flags can be saved on the stack with a PUSH instruction, or restored from the stack with a POP instruction. The two sets of flag registers are paired with the two A accumulators; the current pair is toggled by the EX AF,AF' instruction.

Operating Modes

The multiple operating modes of the processor allows Z80 and Z180 code to be run without change in *virtual Z80* or *virtual Z180* partitions, in the same application with new code that takes advantage of the eZ80's 16-Mbyte linear addressing space and enhanced instruction set.

These operating modes are governed by four factors:

- A state bit called Address and Data Long (ADL)
- Another state bit called *mixed ADL*
- An 8-bit register called MBASE, and
- The state of the eZ80's 80180-compatible Memory Management Unit (MMU)

Native Z80 Mode. ADL, mixed ADL, and MBASE reset to 0, and the MMU resets to an inactive state. In this Native Z80 state, the programming model includes 16-bit registers and addresses, and a 64 KB memory space at the start of the eZ80's potential 16-Mbyte memory space.

Virtual Z80 Mode. If ADL is cleared, the MMU is not enabled, but MBASE contains a non-zero value, the programming model still includes 16-bit registers and a 64 KB memory space, but this space is relocated by MBASE. In this Virtual Z80 mode, several tasks can each have their own Z80 partition.

Native Z180 Mode. If ADL is cleared, MBASE contains zero, and the MMU is active, the programming model is fully Z80180-compatible. The model includes 16-bit registers and a 64 KB *logical* memory addressing space, but the MMU translates these logical addresses to 20-bit *physical* addresses. The 64 KB logical address space can be divided into one to three *areas*, two of which can be relocated anywhere within the first 1 MB of the eZ80's potential 16-Mbyte memory space.

Virtual Z180 Mode. If ADL is cleared, the MMU is active, and MBASE contains a non-zero value, the MMU handles mapping within a 1M byte virtual physical address space that is relocated by MBASE. In this Virtual Z180 mode, several tasks can each have their own Z180 partition.

ADL Mode. If ADL is set, neither the MMU nor MBASE has any effect on memory addressing. In this mode, the PC, BC, DE, HL, IX and IY registers are expanded from 16 to 24 bits, and a 24-bit Stack Pointer Long (SPL) register replaces the 16-bit Stack Pointer Short (SPS) register that is used in the other modes. When the processor fetches an instruction that includes a 16-bit address or immediate datum in the other modes, it automatically fetches a 24-bit address or datum. Code that operates in ADL mode must be generated by an eZ80-compatible compiler or assembler that generates such instructions.

Mode Switching. The eZ80 switches between ADL mode and any of the other modes only as part of a specially-prefixed CALL, JP, RET, or RST instruction, or an interrupt or trap operation. The MBASE register can be changed only in ADL mode. The MMU can be programmed in any mode, but in a non-ADL mode software must take care not to affect its Program Counter when programming the MMU.

Interrupt and Traps. Applications that operate only in Native Z80 mode, ADL mode, or Native Z180 mode with Common Bank 0 always enabled, are relatively simple with respect to interrupts and traps. In these modes, memory always starts at the start of the eZ80's potential 16-Mbyte memory space, and the interrupt and trap locations are never mapped.

However, applications that switch between modes, or operate in Virtual Z80, Virtual Z180, or Native Z180 mode with Common Bank 0 disabled, simplify interrupts and trap handling by executing a STMIX instruction to set the mixed ADL bit.

If the mixed ADL bit is 1, interrupts and instruction traps stack the ADL state as well as the PC, and enter ADL mode in the first 64K bytes of the eZ80's potential 16M byte memory space.

I/O Space

A separate I/O space includes on-chip and off-chip peripheral devices. On the Z80, I/O space included 8-bit addresses and 256 bytes. All Z180 processors, and the eZ80, feature an expanded I/O space with 16-bit addresses and 64K bytes. The eZ80 includes a few on-chip peripherals in I/O space, which can be augmented by external peripherals.

Other Processor Control Registers

Interrupt High Address (I). The contents of this register are used as the eight high-order address bits, when the processor fetches the address of an interrupt service routine from memory, for an interrupt from the $\overline{\texttt{INT1}}$ or $\overline{\texttt{INT2}}$ pin, or from an on-chip peripheral. The `I` register points at a table of interrupt service routine addresses, that starts at a 256-byte boundary in the 64K-byte logical address space. The `I` register resets to 0, and can be read or written by the dedicated instructions LD A,I and LD I,A.

R Counter (R). On the Z8018x processors family, this register contains a count of executed fetch cycles. R resets to 0, and can be read or written by the dedicated instructions LD A,R and LD R,A.

Illegal Instruction Traps

Like most processors, the defined instruction set for the Z8018x family does not fully cover all possible sequences of binary values. The Op Code maps, which begin on page 43, include numerous blank cells. These represent Op Code sequences for which no operation is defined, and are commonly called illegal instructions.

When an eZ80 or other Z8018x processor fetches one of these sequences, it performs a Trap sequence as follows:

1. The processor sets the TRAP bit in the Interrupt/Trap Control register.
2. If the processor detected the condition while fetching the second byte of the instruction, it clears the UFO bit in the Interrupt/Trap Control register. If the processor detected the condition while fetching the third byte, the processor sets UFO.
3. The processor decrements the Stack Pointer (SP) by two and stores the 16-bit logical address from PC, in memory at the new SP address. This address points to the last byte of the illegal Op Code sequence.
4. The processor then clears PC and resumes execution at logical address 0000.

Trap Handling. The code at logical address 0000 can optionally store the value of SP in memory, and then set SP to an area of memory dedicated to its private stack.

In all cases, the trap-handling routine must store as many registers among AF, BC, DE, HL, IX, and IY as it may use (worst case), by pushing them onto the stack. A general-purpose routine stores all of these registers, those in the alternate set, the value of I and the state of the Interrupt Enable flag.

Next, the Trap-handling code must distinguish among the four events that can bring execution to address 0000:

- A Reset
- A Trap
- A RST 0 instruction
- A program error such as a JUMP to a null pointer.

The code can detect a Trap by reading the Interrupt/Trap Control register (ITC) and checking bit 7 (TRAP). If this bit is 1, a Trap has occurred, and ZiLOG recommends that the Trap-Handling Routine proceed as follows:

- Clear the TRAP bit by writing a 0 to bit 7 of the ITC
- Fetch the PC value stored on the stack
- Examine bit 6 of the ITC (UFO).

- If the UFO bit is 0, decrements the PC value by one, else decrement it by two, so that it points to the start of the illegal instruction.

The next action of the trap-handling routine depends on the application and its stage of development.

Extending the Instruction Set. Core software can use illegal instructions as extensions to the eZ80 instruction set. To accomplish this, the trap handler must fetch and examine each illegal instruction. If an illegal instruction is an extension, the trap handler performs the extended operation that the instruction indicates. It then advances the stacked PC value over the instruction, restores the saved register values, and returns to the next instruction.

Error Message vs. Restart. Except for such extended instructions, the trap handling software can either:

- Output an error message and wait for someone to examine the situation and restart the application, or
- Attempt to restart the application immediately.

The former course is more common in the debugging/development stages of an application, while the latter may be more appropriate in the production/deployment stage. In the latter case, software may log the event for future readout, using an external storage medium or just in memory.

INTERRUPTS

ZiLOG Z80 and Z80180 processors have a rich legacy of sophisticated interrupt capabilities. The eZ80 includes aspects of both families' interrupt characteristics.

Interrupt Resources in the eZ80

IEF1 and IEF2. These bits are internal to the processor and can only be affected and manipulated by certain specific events:

- Reset clears IEF1 and IEF2
- EI instructions set IEF1 and IEF2
- DI instructions clear IEF1 and IEF2
- An NMI sequence copies IEF1 to IEF2, then clears IEF1
- A maskable interrupt clears IEF1 and IEF2
- An LD A,I or LD A,R instruction copies IEF2 to the P/V flag
- An RETN instruction copies IEF2 to IEF1

When IEF1 is 1, $\overline{\text{RESET}}$ and $\overline{\text{BUSREQ}}$ are both High, and falling edge has occurred on $\overline{\text{NMI}}$, the eZ80 checks for maskable interrupt requests from external pins and on-chip peripherals, as it completes each instruction, or each instruction iteration for HALT, the block I/O instructions, block move instructions, and block scan instructions.

The I Register. The eZ80 uses the contents of this register as A15–8 of the logical address for fetching interrupt service routine addresses from memory, and in response to interrupt requests from internal peripherals.

Nonmaskable Interrupt (NMI)

The eZ80 latches falling edges on the $\overline{\text{NMI}}$ pin. Only a Low on $\overline{\text{RESET}}$ or on $\overline{\text{BUSREQ}}$ takes precedence over $\overline{\text{NMI}}$. Unless $\overline{\text{RESET}}$ or $\overline{\text{BUSREQ}}$ is Low, the eZ80 checks for a falling edge on $\overline{\text{NMI}}$ as it completes each instruction (each instruction iteration of `HALT`, the block I/O instructions, block move instructions, and block scan instructions), and performs an NMI sequence if a falling edge has occurred.

An NMI sequence includes four steps. The processor:

1. Copies the state of the IEF1 bit to IEF2.
2. Clears `IEF1` to prevent maskable interrupts.
3. Decrements SP by 2, and stores the logical address in the `PC` in memory at the new address in SP. For most interrupts, this address is the address of the instruction the processor would have executed next, if no interrupt had occurred. If the processor was stopped by `HALT` or `SLP`, the value is the address of the next instruction. In the event of an incomplete block transfer, block scan, or block I/O instruction, it is the address of the instruction.
4. Loads 0066H into `PC`, and resumes execution from that logical address.

NMI Handling. NMI routines fall into two categories, based on whether the external hardware that drives $\overline{\text{NMI}}$ is capable of producing another falling edge on the pin, before the NMI service routine has completed its execution and returned to the interrupted process. The case when another falling edge cannot be produced is called *Single Edge Guaranteed.* The case when it is possible to produce another falling edge is called *Repeated Edge Possible*. Debug monitors, which display the state of the interrupted process, fall into the Repeated Edge category.

Single Edge Guaranteed. An NMI routine in this category is similar to other interrupt service routines. This routine has the option of storing the contents of `SP` in memory and loading `SP` with the address of a memory area that is dedicated for its stack. In any case it must store as many of the registers as it may use during its execution (worst case).

Repeated Edge Possible. An NMI routine in this category start with a `PUSH AF` instruction, then load A from a dedicated location in memory that indicates whether the interrupted process is the NMI routine. If this location indicates that the location is the NMI process, the routine immediately must perform a `POP AF` and then an `RETN` instruction, to return to its former execution.

If the `in NMI` location is cleared, software must set it. Then, if the NMI routine does either of the following:

- A DI instruction in a *Save The Registers* routine that it shares with other means of entry, or

- Displays the I register or the interrupt-enable state of the interrupted process, and allows a user/programmer to change these (in essence, a debug monitor)

the NMI must perform LD A,I and PUSH AF instructions. These instructions store the I register at the address in SP plus 1, and the interrupt enabled state (IEF2) in the
P/V flag and in bit 2 of the memory location pointed to by SP.

If the NMI routine uses a common *Save The Registers* subroutine that it shares with other entry points, the save subroutine can perform a DI instruction to prevent interruption by maskable interrupts.

The NMI routine has the option to store the SP value in a dedicated location in memory, and load SP with the address of a dedicated NMI stack area.

In any case, the NMI routine must PUSH as many other registers as it will use (worst case). A debug monitor typically pushes all registers in both banks, so that it can display them.

Exiting The NMI Routine. On completion of its processing, an NMI routine must restore the saved registers. If the routine uses its own stack area, the routine then restores the SP value of the interrupted process. If the routine set an in NMI memory location on the way in, the routine clears this location.

NMI routines that did *not* save the I register and IEF2 state at the start, can conclude with POP AF and RETN instructions. RETN copies the state of IEF2 back into IEF1, restoring the interrupt enable state of the interrupted process.

NMI routines that saved I and IEF2 at the start, must conclude with a POP AF for the saved I register and IEF2 bit, then, an LD I,A, followed by a JP V to a POP AF, EI, RET sequence. The JP instruction is followed by LD I,A, POP AF, and RET instructions.

INT0

INT0 Modes. The eZ80 can handle interrupts requested by a device on the INT0 pin, in any of three ways called modes 0, 1, or 2.

The special instructions IM 0, IM 1, and IM 2 select among these three modes. Reset selects mode 0.

INT0 Processor Response. The eZ80 performs an INT0 interrupt sequence at the end of an instruction[1], if all of the following are true:

- INT0 is Low
- Bit 0 of the Interrupt/Trap Control register is 1 to enable INT0
- IEF1 is 1, enabling interrupts in general
- RESET and BUSREQ are both High
- A negative edge on NMI has not been detected

1. Each instruction iteration for HALT, the block I/O, block move, and block scan instructions

When all of these conditions occur simultaneously, the eZ80 responds.

While all $\overline{\text{INT0}}$ acknowledge cycles follow a general pattern, they differ as to what (if anything) the processor does with the data on D7-0, and what it does after the acknowledge cycle. These actions depend on the most recently executed IM instruction (if any), as described in the next three sections.

$\overline{\text{INT0}}$ Mode 0. If no IM instruction has been executed since Reset, or if the most recently executed IM instruction was IM 0, the eZ80 performs an $\overline{\text{INT0}}$ sequence as follows:

1. It clears IEF1 and IEF2, preventing further interrupts
2. It drives $\overline{\text{INSTRD}}$ Low
3. It waits several clock cycles
4. It drives $\overline{\text{IORQ}}$ Low. Simultaneous lows on $\overline{\text{INSTRD}}$ and $\overline{\text{IORQ}}$ indicate an $\overline{\text{INT0}}$ interrupt acknowledge cycle. In response to this condition, the highest-priority peripheral that is requesting an interrupt places an 8-bit value on the D7-0 data bus.
5. It samples $\overline{\text{WAIT}}$, and waits until it is High.
6. It samples D7-0 and interprets the value as an instruction Op Code. In this mode, the vector registers of all ZiLOG daisy-chainable peripherals must be programmed to provide one of the RST Op Codes C7, CF, D7, DF, E7, EF, F7, or FFH.

Notes:

1. Read RST as *Restart*.
2. The eZ80 does not automatically stack the contents of the program counter during an INT0 Mode 0 interrupt sequence. This means that the only other Op Code that a peripheral can return (assuming the interrupted process is to be restarted) is a CALL instruction DCH. Intel 808x-family interrupt controllers can return a three-byte CALL instruction, but ZiLOG peripherals cannot.

7. It terminates the cycle by driving $\overline{\text{INSTRD}}$ High, then $\overline{\text{IORQ}}$ High.
8. If the Op Code is CALL, the processor fetches two more bytes to complete the instruction.
9. The processor decrements SP by 2 and stores the contents of PC in memory at the new address in SP. Typically, this value is the address of the instruction the processor would have executed next, if no interrupt had occurred. If the processor was stopped by HALT or SLP, this value is the address of the next instruction. For an incomplete block transfer, block scan, or block I/O instruction, this value is the address of the instruction.
10. If the Op Code was RST, the processor resumes execution at logical address 0000, 0008, 0010, 0018, 0020, 0028, 0030, or 0038H. If the Op Code was CALL, it resumes execution at the logical address fetched in step 8.

In mode 0, each peripheral connected to INT0 must feature a register, the contents of which it returns on D7-0 when it detects INSTRD and IORQ Low, and it is requesting an interrupt, and the IEI pin is High. Software programs each such register with one of the `RST` Op Codes `C7`, `CF`, `D7`, `DF`, `E7`, `EF`, `F7` or `FFH`.

If a peripheral can replace the low-order bits of this value with a code reflecting the peripheral's status, this feature must be disabled for mode 0 operation.

If the number of devices that can interrupt on INT0 is restricted, each device can have its own `RST` instruction, improving interrupt response time by eliminating the requirement for the interrupt service routine to poll multiple devices.

If multiple devices must share a `RST` instruction, that interrupt service routine must poll these devices in the same priority order that they are arranged on the IEI-IEO daisy chain. Because a ZiLOG peripheral sets its `IUS` bit when:

- It detects `INSTRD` and `IORQ` Low
- It is requesting an interrupt, and
- Its `IEI` pin is High.

To insure correct operation of the daisy chain, the polling process must lead to:

- Servicing the highest priority requesting device that performed the poll
- Clearing its `IUS` bit either explicitly, or for a Z80 peripheral, by concluding the ISR with a `RETI` instruction.

INT0 Mode 1. If the most recently executed IM instruction was IM 1, the eZ80 performs an `INT0` sequence as follows:

1. It clears `IEF1` and `IEF2`, preventing further interrupts
2. It drives `INSTRD` Low
3. It waits several clock cycles
4. It drives `IORQ` Low. Simultaneous Lows on `INSTRD` and `IORQ` indicate an `INT0` interrupt acknowledge cycle
5. It samples `WAIT`, and waits until it is High.
6. It terminates the cycle by driving `INSTRD` High, then `IORQ` High.
7. It ignores the data on D7-0 and substitutes the value `FFH,` which is `RST` 38.
8. It decrements SP by 2, and stores the contents of PC in memory at the new logical address in SP. Typically, this address is the address of the instruction the processor would have executed next, if no interrupt had occurred. If the processor was stopped by `HALT` or `SLP`, this value is the address of the next instruction. For an incomplete block transfer, block scan, or block I/O instruction, this value is the address of the instruction.
9. It loads `0038H` into PC, and resumes instruction execution from that logical address.

In mode 1, the interrupt service routine must poll all of the devices connected to INT0, to determine which device generated the interrupt. If any ZiLOG peripherals can request an interrupt, this polling must be performed in the same priority order that the devices are arranged on the IEI-IEO daisy chain. This order is required because a ZiLOG peripheral sets its IUS bit when:

- It detects INSTRD and IORQ Low
- It is requesting an interrupt, and
- Its IEI pin is High.

To ensure correct operation of the daisy chain the polling process must lead to:

- Servicing the device that requested the interrupt
- Clearing the device's IUS bit either explicitly, or for a Z80 peripheral, by concluding the ISR with a RETI instruction.

The best way to ensure this requirement is by actually polling the IUS bits, for devices that allow their IUS bits to be read.

INT0 Mode 2. If the most recently executed IM instruction was IM 2, the eZ80 performs an INT0 sequence as follows:

1. It clears IEF1 and IEF2, preventing further interrupts
2. It drives INSTRD Low
3. It waits several clock cycles
4. It drives IORQ Low. Simultaneous lows on INSTRD and IORQ indicate an INT0 interrupt acknowledge cycle. In response to this condition, the highest-priority peripheral that is requesting an interrupt places an 8-bit value on the D7-0 data bus.
5. It samples WAIT, and waits until it is High.
6. It captures the data from D7-0. This byte must have D0 Low/0 for proper operation.
7. It terminates the cycle by driving INSTRD High, then IORQ High.
8. It decrements SP by 2, and stores the contents of PC in memory at the new logical address in SP. Typically, this value is the address of the instruction the processor would have executed next, if no interrupt had occurred. If the processor was stopped by HALT or SLP, this value is the address of the next instruction. For an incomplete block transfer, block scan, or block I/O instruction, this value is the address of the instruction.
9. It places the contents of the I register on A15-8, the value captured in step 7 on A7-0, and fetches the Less Significant (LS) byte of an interrupt service routine address from memory at that address.
10. It drives A0 to go High/1, and fetches the More Significant (MS) byte of the interrupt service routine address from memory at that address.
11. It esumes execution at the logical address fetched in steps 9-10.

In mode 2, each peripheral connected to $\overline{\text{INT0}}$ must have an Interrupt Vector Register, the contents of which it returns when:

- The peripheral detects $\overline{\text{INSTRD}}$ and $\overline{\text{IORQ}}$ Low
- The peripheral is requesting an interrupt, and
- Its IEI pin is High.

User software can program each Interrupt Vector register with any even binary value.

If a peripheral can replace the low-order bits of this value with a code reflecting its status, this feature can be enabled in mode 2 (in which case the peripheral *occupies more than one slot* in the interrupt vector table). These *Vector includes status* features improve interrupt response time, reducing the amount of status-polling that the interrupt service routine must do, to identify the exact cause of the interrupt.

Interrupt Handling. Any Interrupt Service Routine (ISR) may save the contents of SP in memory, and loads SP with the address of a memory area that is dedicated to its stack. Most interrupt service routines do not include these steps.

An $\overline{\text{INT0}}$ ISR must save the contents of the registers it uses (worst case), using PUSH and/or EX AF,AF' and EXX instructions.

If the application includes a mechanism for allowing nested interrupts, the ISR can begin as specified by that mechanism, leading to an IE instruction that allows the ISR to be interrupted by other interrupts. Most applications do not include these steps.

Next, the ISR must read status registers from each device that can request an interrupt on $\overline{\text{INT0}}$, to identify the cause of the interrupt. The ISR must handle each interrupting device according to this status, and the device and application requirements.

Many ISRs read data from interrupting device(s), or write data to interrupting device(s). In addition, the ISRs can write registers in interrupting device, to modify the device(s) mode, status, or operation.

When interrupt processing is complete, the ISR may end either of two ways. If nested interrupts were allowed, the ISR ends as specified by the nesting mechanism. If nested interrupts were not allowed, the ISR must restore the saved registers and conclude with EI and RET instructions.

NOTE: the Z80 and Z80180 instruction sets include an RETI instruction, that is used for servicing Z80 peripherals. Because the eZ80 does not include Z80 peripherals, nor does it allow them to be connected externally, there is no reason to ever conclude a eZ80 ISR with an RETI. RET is both shorter and faster than RETI, and has the same function.

MEMORY

The eZ80 provides several address-generation modes:

Native Z80 mode. The total memory address space is the first 64K bytes of the overall eZ80 memory space. Neither the Z80180-compatible Memory Management Unit (MMU) nor the Memory Base (MBASE) register has any effect on addressing.

Virtual Z80 mode. The memory address space can be any 64 KB in the overall 16M byte eZ80 memory space, under control of the MBASE register. The MMU has no effect on memory addressing.

Native Z180 mode. The memory address space is the first 1M bytes of the overall eZ80 memory space, under control of the Z180-compatible MMU. MBASE has no effect on memory addressing.

Virtual Z180 mode. This mode allows the memory address space to be any 1 MB of the overall 16 MB eZ80 memory space, under control of the MBASE register. The MMU operates within the selected 1 MB space.

Address and Data Long (ADL) mode. This mode allows programs compiled or assembled for the eZ80 to operate in a 16M byte linear address space. In this mode, the 16-bit registers PC, BC, DE, HL, IX, and IY expand to 24 bits, as does the width of the ALU. The processor automatically fetches an additional byte of address or immediate data in those instructions that contain a 16-bit address or datum in other modes.

Prefix-override bytes allow any instruction to operate as in ADL mode in one of the first four modes, or to use 16 bits MMU or MBASE addressing in ADL mode.

Addressing Modes

Instructions may specify a memory address in several ways. eZ80 addressing modes include:

Relative Addressing. `JR` and `DJNZ` instructions include a signed 8-bit displacement that specifies a range of addresses –126 to +129 from the Op Code, to which program control can be transferred.

Direct Addressing. Instructions include a 16-bit or 24-bit logical address, depending on the ADL mode bit.

Register Indirect Addressing. The address is taken from one of the register pairs `BC`, `DE` or `HL`.

Indexed Addressing. In this mode, instructions include an 8-bit signed displacement from the address in an index register, IX or IY.

Other contexts in which memory is accessed include instruction fetching, interrupts, DMA operations, and cycles generated by external masters while $\overline{\text{BUSACK}}$ is Low.

Memory Management Unit (MMU)

The eZ80 includes an 80180-compatible Memory Management Unit to enable programs written for an 8018x family processor to be run without change. For new code, the 24-bit linear address mode is far more straightforward and easier to use.

The 16-bit address used by software are called logical addresses. When the MMU is enabled it translates these 16-bit logical addresses into 20-bit physical addresses, as part of all memory accesses performed by the processor. The MMU has no effect on accesses performed by the DMA channels, which include 20-bit address registers. The MMU also has no effect on addresses in I/O space, which always have A23–16 all 0.

The MMU resets to a state in which it has no effect on addresses in processor cycles, passing A15–0 through without change and keeping A23–16 all 0. If an application needs 64 KB of memory or less, the MMU need not be used.

Even when the MMU has been programmed to perform active address translation, it passes A11–0 from the logical to the physical address. In other words, it manages memory in 4 KB blocks.

The section “MMU Registers” on page 20, details the registers associated with the MMU.

MMU Operation. The MMU compares bits 15–12 of each logical address to two 4-bit fields in its Common/Base Address Register (`CBAR`). These comparisons are unsigned.

If bits 15–12 of a logical address are less than the value in bits 3–0 of the `CBAR`, the MMU maps the address to Common Area 0. For these addresses, the MMU passes bits 15–12 to the `A15-12` pins unchanged, and sets A23–16 to 0.

If bits 15–12 of a logical address are greater than or equal to the value in bits 3–0 of the `CBAR`, but are less than the value in bits 7–4 of the `CBAR`, the MMU maps the address to the Bank Area. For these addresses, the MMU adds the value in its 8-bit Bank Base Register (`BBR`) to bits 15–12 of the logical address, and outputs the 8-bit sum on A19–12, and sets A23-20 to 0.

If bits 15–12 of a logical address are greater than or equal to the value in bits 7–4 of the `CBAR`, the MMU maps the address to Common Area 1. For these addresses, the MMU adds the value in its 8-bit Common Base Register (`CBR`) to bits 15–12 of the logical address, outputs the 8-bit sum on A19–12, and sets A23-20 to 0.

NOTE: The value in bits 7–4 of the `CBAR` must never be less than the value in bits 3–0 of the `CBAR`.

MMU Configurations. In the general case, the MMU divides the 64 KB logical memory space into three parts. Common Area 0 is located at the start of the 1 MB physical address space. The Bank Area and Common Area 1 are relocatable to other parts of the physical address space under control of the Bank Base Register and Common Base Register, respectively.

Certain combinations of values in the CBAR result in the logical address space being divided into fewer active areas:

- If the CBAR contains 0, all logical addresses fall into Common Area 1, and are relocated to a contiguous 65KB area starting at the address in the CBR times 4096.
- If CBAR3–0 are 0 but CBAR7–4 are non-zero, the Bank Area and Common Area 1 are active. Logical addresses less than `(CBAR7–4)*4096` are relocated by the Bank Base Register while other addresses are relocated by the Common Base Register.
- If CBAR7–4 and CBAR3–0 are equal and non-zero, Common Area 0 and Common Area 1 are active. Logical addresses less than `(CBAR3–0)*4096` are not relocated and map to the start of physical memory. Other addresses are relocated by the Common Base Register.

The MMU After Reset. Because the CBAR resets to `11110000`, logical addresses `0000–EFFFH` are in the Bank Area and `F000–FFFFH` are in Common Area 1 after Reset. But since the `BBR` and `CBR` both reset to zero, the MMU passes all logical addresses through without change, with A23–16 all 0.

INPUT/OUTPUT

The eZ80 includes an I/O space that is distinct from memory space. I/O space is accessed by means of `IN` and `OUT` instructions rather than `LD`, `PUSH`, `POP`, and other instructions that access memory space. The MMU passes addresses in I/O space through without change; these addresses always have A23–16 all 0.

I/O Instructions

The original Z80 featured a 256-byte I/O space. The following instructions are specific to the Z80's 256-byte I/O space, and must only be used on the eZ80 to address external I/O devices that do not decode A15–8:

```
OUT  (port), A
IND
INDR
INI
INIR
OTDR
OTIR
OUTD
OUTI
```

The following instructions ensure that A15–8 are all 0 and can be used to access the eZ80's on-chip I/O registers, as well as external devices that decode A15–8 as all 0s:

```
IN0   r, (port)
OUT0  (port), r
OTDM
OTDMR
OTIM
OTIMR
```

The following instructions drive A15–0 from the BC register pair and can be used to access the full 64 KB I/O space:

```
IN   r, (C)
OUT  (C), r
```

The following instruction can access the entire 64 KB I/O space, by pre-loading bits 15–8 of the address into A. (This step is unnecessary for external devices that do not decode A15-8.)

```
IN   A, (port)
```

CLOCK CIRCUITS

The eZ80 requires a logic-level clock on its CLK pin. This signal must be free of overshoot or ringing and must make continuous, monotonic, and rapid transitions in both directions.

RESET CONDITIONS

The effects of Reset on each of the registers in I/O space is described in Tables 2–4 in “I/O Registers”. Among processor registers, the following registers and state bits are cleared to 0: ADL, Mixed ADL, MBASE, PC, SP, I, IEF1, IEF2, R, and F. The following are not changed by Reset: A, B, C, D, E, H, L, IX, and IY.

I/O REGISTERS

Preceding sections describe the processor registers and the eZ80’s programming model. This section describes the registers in I/O space that control the operation of the overall device and its on-chip peripherals. Register addresses that do not appear in this table are not used.

REGISTERS SUMMARY

Register Name	Addr (hex)	Register Name	Addr (hex)
Common Base Register	38	Bank Base Register	39
Common/Bank Area Register	3A		

MMU REGISTERS

See the Memory Management Unit section, starting on page 18, for additional register information.

TABLE 2. COMMON BASE REGISTER (0038H) CBR

Bit	7	6	5	4	3	2	1	0
Bit/Field	Base of Common Area 1							
R/W	R/W							
Reset	0	0	0	0	0	0	0	0
Note: R = Read W = Write X = Indeterminate								

Bit Position	Bit/Field	R/W	Value	Description
7-0	Common 1 Area Base	R/W		If the comparison of bits 15-12 of a logical address indicates that the address is in Common Area 1, this value (shifted left 12 bits, times 4096) is added to the logical address to form the physical address.

TABLE 3. BANK BASE REGISTER (0039H) BBR

Bit	7	6	5	4	3	2	1	0
Bit/Field	Base of Bank Area							
R/W	R/W							
Reset	0	0	0	0	0	0	0	0
Note: R = Read W = Write X = Indeterminate								

Bit Position	Bit/Field	R/W	Value	Description
7-0	Bank Area Base	R/W		If the comparison of bits 15–12 of a logical address indicates that the address is in the Bank Area, this value (shifted left 12 bits, times 4096) is added to the logical address to form the physical address.

TABLE 4. COMMON/BANK AREA REGISTER (003AH) CBAR

Bit	7	6	5	4	3	2	1	0
Bit/Field	Bank/Common 1 Boundary				Common 0/Bank Boundary			
R/W	R/W				R/W			
Reset	1	1	1	1	0	0	0	0
Note: R = Read W = Write X = Indeterminate								

Bit Position	Bit/Field	R/W	Value	Description
7-4	Bank/ Common 1 Boundary	R/W		If bits 15–12 of a logical address are greater than or equal to this value, the address is in Common Area 1.
3-0	Common 0/ Bank Boundary	R/W		If bits 15–12 of a logical address are less than this value, the address is in Common Area 0.

Note: If bits 3-0 of this reg $\leq$ bits 15-12 of a logical address < bits 7-4 of this reg, the address is in the Bank Area. Do not program this register so that bits 3-0 > bits 7-4. All comparisons are unsigned.

INSTRUCTION SET

The eZ80 is descended from the ZiLOG Z80. Its 8-bit data bus and 24-bit address space fit well into a wide variety of mid-range embedded processing applications. This processor provides significantly more computing power than a microcontroller, at a fraction of the system cost of a larger microprocessor.

Instructions and features that are new to the eZ80 are denoted by a dagger (†). Instructions that exist in the Z80 but are undocumented, do not exist in the Z8018x family, and are implemented and acknowledged in the eZ80, are denoted by a double dagger (‡).

CLASSES OF INSTRUCTIONS

TABLE 5. LOAD INSTRUCTIONS

Mnemonic	Operands	Instruction
LD	dst,src	Load
LEA	qq,IX/Y ± d	Load Effective Address †
PEA	IX/Y ± d	Push Effective Address†
POP	dst	Pop
PUSH	src	Push
† Instructions and features that are new to the eZ80.		

TABLE 6. ARITHMETIC INSTRUCTIONS

Mnemonic	Operands	Instruction
ADC	dst,src	Add with Carry
ADD	dst,src	Add
CP	A,src	Compare
CPD(R)		Block Scan, decrementing (and Repeat)
CPI(R)		Block Scan, incrementing (and Repeat)
DAA		Decimal Adjust Accumulator
DEC	dst	Decrement
INC	dst	Increment
MLT	rr	Multiply
NEG		Negate Accumulator
SBC	dst,src	Subtract with Carry
SUB	A,src	Subtract

TABLE 7. LOGICAL INSTRUCTIONS

Mnemonic	Operands	Instruction
AND	A,src	Logical AND
CPL		Complement accumulator
OR	A,src	Logical OR
TST	A,src	Test accumulator
XOR	A,src	Logical Exclusive OR

TABLE 8. EXCHANGE INSTRUCTIONS

Mnemonic	Operands	Instruction
EX	AF,AF’	Exchange Accumulator and Flags
EX	DE,HL	Exchange DE and HL
EX	(SP),rr	Exchange register and top of stack
EXX		Exchange register banks

TABLE 9. PROGRAM CONTROL INSTRUCTIONS

Mnemonic	Operands	Instruction
CALL	cc,dst	Conditional Call
CALL	dst	Call
DJNZ	dst	Decrement and Jump if Non-Zero
JP	cc,dst	Conditional Jump
JP	dst	Jump
JR	cc′,dst	Conditional Jump Relative
JR	dst	Jump Relative
RET	cc	Conditional Return
RET		Return
RETI		Return from Interrupt
RETN		Return from Nonmaskable interrupt
RST	dst	Restart

TABLE 10. BIT MANIPULATION INSTRUCTIONS

Mnemonic	Operands	Instruction
BIT	n,src	Bit test
RES	n,dst	Reset bit
SET	n,dst	Set bit

TABLE 11. BLOCK TRANSFER INSTRUCTIONS

Mnemonic	Operands	Instruction
LDD(R)		Block Move, decrementing (and Repeat)
LDI(R)		Block Move, incrementing (and Repeat)

TABLE 12. ROTATE AND SHIFT INSTRUCTIONS

Mnemonic	Operands	Instruction
RL	dst	Rotate Left
RLA		Rotate Left Accumulator
RLC	dst	Rotate Left Circular
RLCA		Rotate Left Circular Accumulator
RLD		Rotate Left Decimal
RR	dst	Rotate Right
RRA		Rotate Right Accumulator
RRC	dst	Rotate Right Circular
RRCA		Rotate Right Circular Accumulator
RRD		Rotate Right Decimal
SLA	dst	Shift Left
SRA	dst	Shift Right Arithmetic
SRL	dst	Shift Right Logical

TABLE 13. INPUT/OUTPUT INSTRUCTIONS

Mnemonic	Operands	Instruction
IN	A, (n)	Input to A from port n
IN	r, (C)	Input to register from port in BC
IN0	r, (n)	Input to r from port n in page 0
IND(R)		Block Input, decrement HL (and Repeat)
IND2(R)		Block Input, decrement both (and Repeat)†
INDM(R)		Block Input, page 0, decrement both (and Repeat)†
INI(R)		Block Input, increment HL (and Repeat)
INI2(R)		Block Input, decrement both (and Repeat)†
INIM(R)		Block Input, page 0, increment both (and Repeat)†
OTDM(R)		Block Output, page 0, decrement both (and Repeat)
OTIM(R)		Block Output, page 0, increment both (and Repeat)
OUT	(n), A	Output from A to port n
OUT	(C), r	Output from register to port in BC
OUT0	(n), r	Output from register to port n in page 0
OUTD (OTDR)		Block Output, decrement HL (and Repeat)
OUTD2 (OTD2R)		Block Output, decrement both (and Repeat)†
OUTI (OTIR)		Block Output, increment HL (and Repeat)
OUTI2 (OTI2R)		Block Output. decrement both (and Repeat)†
TSTIO	n	Test port (0,C) under mask
† Instructions and features that are new to the eZ80.		

TABLE 14. PROCESSOR CONTROL INSTRUCTIONS

Mnemonic	Operands	Instruction
CCF		Complement Carry Flag
DI		Disable Interrupts
EI		Enable Interrupts
HALT		Halt
IM	0/1/2	Interrupt Mode
NOP		No Operation
RSMIX		Reset Mix Flag†
SCF		Set Carry Flag
SLP		Sleep
STMIX		Set Mix Flag†
† Instructions and features that are new to the eZ80.		

PROCESSOR FLAGS

Table 15 shows the Flag register. Bits in this register are set and cleared by certain instructions as described in the eZ80 User Manual. Some of the Flags are tested by conditional `JR`, `JP`, `CALL`, and `RET` instructions, and some are used by subsequent instructions such as `ADC`, `SBC`, and `DAA`. The Flags can also be pushed and popped with accumulator A.

TABLE 15. FLAG REGISTER

Bit	7	6	5	4	3	2	1	0
Name	S	Z	x	HC	x	P/V	N	CF
Reset	0	0	x	0	x	0	0	0
Note: X = Indeterminate								

Bit/ Field	Bit Position	Description
S	7	Sign Flag
Z	6	Zero Flag
	5	reserved
HC	4	Half-carry Flag
	3	reserved
P/V	2	Parity or Overflow Flag
N	1	Add/Subtract Flag
CF	0	Carry Flag

CONDITION CODES

Table 16 shows the codes used in the *Flags Affected* columns of the Instruction Summary Table, Table 19, to indicate how each flag is affected by each type of instruction.

TABLE 16. FLAG SETTINGS DEFINITIONS

Symbol	Definition
0	Cleared to 0
1	Set to 1
*	Set or cleared according to the result of the operation
–	Unaffected
X	Undefined
V	Set if Overflow or Underflow
P	Set if Parity or result is Even
NZ	Set if the count in B or BC is non-zero

Table 17 shows the condition codes that can be used in conditional `JP`, `CALL`, and `RET` instructions in assembly language. A subset of these codes can also be used in `JR` instructions, which are shorter and faster than `JP`s.

TABLE 17. CONDITION CODES

Mnemonic	Definition	Flag Settings	Valid in JR?
C	Carry	CF = 1	Y
NC	No Carry	CF = 0	Y
Z	Zero	Z = 1	Y
NZ	Non-Zero	Z = 0	Y
M	Minus	S = 1	N
P	Positive or zero	S = 0	N
PE	Parity Even	P/V = 1	N
PO	Parity Odd	P/V = 0	N
V	Overflow	P/V = 1	N
NV	No Overflow	P/V = 0	N

ASSEMBLY LANGUAGE SYNTAX

For two-operand instructions, Z80 assembly language syntax puts the destination operand before the source operand.

EXAMPLE: `LD A,(1234)` is a Load instruction, while `LD (1234),A` is a Store instruction.

Past Z80 assemblers allowed the destination operand to be omitted (implicit) if the Op Code mnemonic only allowed one destination operand, for example, `AND L` instead of `AND A,L`. Use of these short forms is discouraged because it is a cause of possible error (the programmer mistakes the implicit destination). But for legacy code, all known Z80 assemblers still accept the short form.

NOTE: The assembly language uses C ambiguously, to designate one of the 8-bit registers as well as a condition code to test the Carry flag. This processor description uses `CF` to designate the Carry flag, and `HC` to designate the Half-Carry flag (as opposed to the 8-bit register `H`)

NOTATION

Table 18 describes other notation used in the Instruction Summary table.

TABLE 18. SYMBOLS

Symbol	Definition
(aa)	(mn), (IX ± d), (IY ± d), (BC), (DE), or (HL).
(BC), (DE), (HL)	The 8-bit contents of memory, at the address pointed to by a register pair. (HL) can also indicate a 16-bit value in memory.†
(IX ± d), (IY ± d)	The 8- or 16†-bit content of memory at the address formed by adding the contents of the index register and the signed displacement d in the instruction.
(mn)	The 8-bit content of memory at the direct address mn
(SP)	The 16-bit contents of memory at the address pointed to by SP, and the next higher address.
± d	Since d is signed, it would be more correct to just write + instead. But we write ± to emphasize that d is signed.
AF	A concatenated with F, with A as the more significant byte
b	A bit number 0–7
cc	A condition code C, NC, Z, NZ, S, M, PE, PV, V, or NV
cc′	A condition code C, NC, Z, or NZ
d	An 8-bit signed displacement –128 to +127
ee	A 16-bit register BC, DE, HL, SP, IX, or IY
IEF1,2	The processor's two Interrupt Enable Flags.
ih	IXH or IYH‡
il	IYH or IYL‡
ir	IXH, IXL, IYH, or IYL‡
m	An 8-bit variable A, B, C, D, E, H, L, (HL), (IX ± d), or (IY ± d)
mn	A 16-bit immediate data value or direct address
n	A 8-bit immediate value or port number, 0–255 or 0–FFH
op1–op2	A range of Op Code values, that includes some of the values between the low and high values. See the Note.
PC	Program Counter
pp	A 16-bit register BC, DE, HL, SP, IX, IY, or AF

† Instructions and features that are new to the eZ80.
‡ Instructions that exist in the Z80 but are undocumented.

TABLE 18. SYMBOLS

Symbol	Definition
q	A, B, C, D, E, H, L, IXH‡, IXL‡, IYH‡, IYL‡, (HL), (IX ± d), or (IY ± d)
qq	A 16-bit register BC, DE, HL, IX, or IY†
r, r′	An 8-bit register A, B, C, D, E, H, or L.
rae	An 8-bit register A, B, C, D, or E
rr	A 16-bit register HL, IX, or IY.
s	A, B, C, D, E, H, L, IXH‡, IXH‡, IYH‡, IYL‡, n, (HL), (IX ± d), or (IY ± d)
SP	Stack Pointer
ss	A 16-bit register BC, DE, HL, or SP
ss_H, ss_L	The more- and less-significant eight bits of a register pair
tt	A 16-bit register like ss, except that the value that designates HL in the ss encoding, here means *same as the destination register HL, IX, or IY*.

† Instructions and features that are new to the eZ80.
‡ Instructions that exist in the Z80 but are undocumented.

NOTE: The symbol – between Op Codes (op1–op2), in the Op Codes column of the Instruction Summary table, indicates all the binary values between the indicated lower and upper limits inclusive, that can be formed by incrementing the set of bits that differ between the lower and upper value.

EXAMPLE: 00–C0 represents 00, 40, 80, and C0, while 40–BF represents all the values in that range.

INSTRUCTION SUMMARY

The following table describes each type or class of instruction, using the notation described in the preceding sections. In cases where the same location acts as both Destination (Dest) and Source code, is centered between the Dest and Source columns (for example, the DEC instruction).The table is sorted by the assembly language mnemonics.

TABLE 19. INSTRUCTION SUMMARY

	Address Mode		Op Code(s) (Hex)	Flags Affected					
Instruction and Operation	**Dest**	**Source**		**S**	**Z**	**HC**	**P/V**	**N**	**CF**
ADC A,s A ← A + s + CF		r	88–8F	*	*	*	V	0	*
		ir	DD/FD 8C–8D						
		n	CE						
		(HL)	8E						
		(IX/Y ± d)	DD/FD 8E						

TABLE 19. INSTRUCTION SUMMARY (CONTINUED)

	Address Mode		Op Code(s) (Hex)	Flags Affected					
Instruction and Operation	**Dest**	**Source**		**S**	**Z**	**HC**	**P/V**	**N**	**CF**
ADC HL,ss HL ← HL + ss + CF			ED 4A–7A	*	*	*	V	0	*
ADD A,s A ← A + s		r	80–87	*	*	*	V	0	*
		ir	DD/FD 84–85						
		n	C6						
		(HL)	86						
		(IX/Y ± d)	DD/FD 86						
ADD rr,tt rr ← rr + tt	HL		09–39	–	–	*	–	0	*
	IX/Y		DD/FD 09–39						
AND A,s A ← A and s		r	A0–A7	*	*	1	P	0	0
		ir	DD/FD A4–A5						
		n	E6						
		(HL)	A6						
		(IX/Y ± d)	DD/FD A6						
BIT b,m Z ← not (bit b of m)		r	CB 40–7F	X	*	1	X	0	–
		(HL)	CB 46–7E						
		(IX/Y ± d)	DD/FD CB d 46–7E						
CALL cc,Mmn IF cc {SP ← SP – 2 (SP) ← PC15-0 if ADL {SPL ← SPL – 1 (SPL) ← PC23-0} if .i16 OR .i24 { SPL ← SPL-1 (SPL) ← ADL ADL ← .i16 ? 0 : 1} PC15-0 ← mn if ADL {PC23-16 ← M}}			C4–FC	–	–	–	–	–	–
CALL Mmn SP ← SP – 2 (SP) ← PC15-0 if ADL {SPL ← SPL – 1 (SPL) ← PC23-0} if .i16 OR .i24 { SPL ← SPL-1 (SPL) ← ADL ADL ← .i16? 0 : 1} PC15-0 ← mn if ADL {PC23-16 ← M}			CD	–	–	–	–	–	–
CCF CF ← not CF			3F	–	–	*	–	0	*

TABLE 19. INSTRUCTION SUMMARY (CONTINUED)

Instruction and Operation	Address Mode Dest	Address Mode Source	Op Code(s) (Hex)	S	Z	HC	P/V	N	CF
CP A,s A – s		r	B8–BF	*	*	*	V	1	*
		ir	DD/FD BC–BD						
		n	FE						
		(HL)	BE						
		(IX/Y ± d)	DD/FD BE						
CPD A – (HL) HL ← HL – 1 BC ← BC – 1			ED A9	*	*	*	NZ	1	–
CPDR repeat {A – (HL) HL ← HL – 1 BC ← BC – 1 } while (not Z and BC! = 0)			ED B9	*	*	*	NZ	1	–
CPI A – (HL) HL ← HL + 1 BC ← BC – 1			ED A1	*	*	*	NZ	1	–
CPIR repeat {A – (HL) HL ← HL + 1 BC ← BC – 1 } while (not Z and BC! = 0)			ED B1	*	*	*	NZ	1	–
CPL A ← not A			2F	–	–	1	–	1	–
DAA A ← decimal adjust (A,F)			27	*	*	*	P	–	*
DEC ee ee ← ee – 1	ss		0B–3B	–	–	–	–	–	–
	IX/Y		DD/FD 2B						
DEC q q ← q – 1	r		05–3D	*	*	*	V	1	–
	ir		DD/FD 25/2D						
	(HL)		35						
	(IX/Y ± d)		DD/FD 35						
DI IEF1,2 ← 0			F3	–	–	–	–	–	–
DJNZ d B← B – 1 if B != 0 {PC ← PC ± d}			10	–	–	–	–	–	–
EI IEF1,2 ← 1			FB	–	–	–	–	–	–

TABLE 19. INSTRUCTION SUMMARY (CONTINUED)

	Address Mode		Op Code(s) (Hex)	Flags Affected					
Instruction and Operation	**Dest**	**Source**		**S**	**Z**	**HC**	**P/V**	**N**	**CF**
EX AF,AF′ AF ↔ AF′			08	*	*	*	*	*	*
EX (SP),rr (SP) ↔ rr	HL		E3	–	–	–	–	–	–
	IX/Y		DD/FD E3						
EXX BC ↔ BC′ DE ↔ DE′ HL ↔ HL′			D9	–	–	–	–	–	–
HALT			76	–	–	–	–	–	–
IM n			ED 40–58	–	–	–	–	–	–
IN A,(n) A ← (n)			DB	–	–	–	–	–	–
IN r,(C) r ← (BC)			ED 40–78	*	*	0	P	0	–
IN0 r,(n) r ← (0,n)			ED 00–38	*	*	0	P	0	–
INC ee ee ← ee + 1	ss		03–33	–	–	–	–	–	–
	IX/Y		DD/FD 23						
INC q q ← q + 1	r		04–3C	*	*	*	V	0	–
	ir		DD/FD 24/2C						
	(HL)		34						
	(IX/Y ± d)		DD/FD 34						
IND (HL) ← (BC B ← B – 1 HL ← HL – 1			ED AA	X	*	X	X	1	–
IND2 † (HL) ← (BC) B ← B – 1 C ← C – 1 HL ← HL – 1			ED 8C	X	*	X	X	1	–
IND2R † do {(HL) ← (BC) B ← B – 1 C ← C – 1 HL ← HL – 1 } while B != 0			ED 9C	X	1	X	X	1	–
INDM † (HL) ← (0,C) B ← B – 1 C ← C – 1 HL ← HL – 1			ED 8A	*	*	*	P	*	*

TABLE 19. INSTRUCTION SUMMARY (CONTINUED)

Instruction and Operation	Address Mode Dest	Address Mode Source	Op Code(s) (Hex)	S	Z	HC	P/V	N	CF
INDMR † do {(HL) ← (0,C) B ← B – 1 C ← C – 1 HL ← HL – 1 } while B != 0			ED 9A	0	1	0	1	*	0
INDR do {(HL) ← (BC) B ← B – 1 HL ← HL – 1 } while B != 0			ED BA	X	1	X	X	1	–
INI (HL) ← (BC) B ← B – 1 HL ← HL + 1			ED A2	X	*	X	X	1	–
INI2 † (HL) ← (BC) B ← B – 1 C ← C + 1 HL ← HL + 1			ED 84	X	*	X	X	1	–
INI2R † do {(HL) ← (BC) B ← B – 1 C ← C + 1 HL ← HL + 1 } while B != 0			ED 94	X	1	X	X	1	–
INIM † (HL) ← (0,C) B ← B – 1 C ← C + 1 HL ← HL + 1			ED 82	*	*	*	P	*	*
INIMR † do {(HL) ← (0,C) B ← B – 1 C ← C + 1 HL ← HL + 1 } while B != 0			ED 92	0	1	0	1	*	0
INIR do {(HL) ← (BC) B ← B – 1 HL ← HL + 1 } while B != 0			ED B2	X	1	X	X	1	–

TABLE 19. INSTRUCTION SUMMARY (CONTINUED)

Instruction and Operation	Address Mode Dest	Address Mode Source	Op Code(s) (Hex)	S	Z	HC	P/V	N	CF
JP (rr) PC ← rr IF .i16 {ADL ← 0} ELIF .i32 {ADL ← 1}		(HL)	E9	–	–	–	–	–	–
		(IX/Y)	DD/FD E9						
JP cc,Mmn if cc { IF .i16 {ADL ← 0} ELIF .i32 {ADL ← 1} PC ← mn IF ADL {PC23-16 ← M}}			C2–FA	–	–	–	–	–	–
JP Mmn IF .i16 {ADL ← 0} ELIF .i32 {ADL ← 1} PC15-0 ← mn IF ADL {PC23-16 ← M}			C3	–	–	–	–	–	–
JR cc′,d if cc′ {PC ← PC ± d}			10–38	–	–	–	–	–	–
JR d PC ← PC ± d			18	–	–	–	–	–	–
LD (aa),A (aa) ← A	(BC)		02	–	–	–	–	–	–
	(DE)		12						
	(HL)		77						
	(mn)		32						
	(IX/Y ± d)		DD/FD 77						
LD (mn),ee (mn) ← ee		HL	22	–	–	–	–	–	–
		ss	ED 43–73						
		IX/Y	DD/FD 22						
LD (HL),qq † (HL) ← qq		BC,DE,H L	ED 0F–2F	–	–	–	–	–	–
		IX	ED 3F						
		IY	ED 3E						
LD (IX/Y ± d),qq † (IX/Y ± d) ← qq		BC,DE,H L	DD/FD 0F–2F	–	–	–	–	–	–
		same I	DD/FD 3F						
		other I	DD/FD 3E						
LD A,(aa) A ← (aa)		(BC)	0A	–	–	–	–	–	–
		(DE)	1A						
		(HL)	7E						
		(mn)	3A						
		(IX/Y ± d)	DD/FD 7E						

TABLE 19. INSTRUCTION SUMMARY (CONTINUED)

Instruction and Operation	Address Mode Dest	Address Mode Source	Op Code(s) (Hex)	S	Z	HC	P/V	N	CF
LD A,I A ← I			ED 57	*	*	0	IEF2	0	–
LD A,MB † if ADL, A ← MB			ED 6E	–	–	–	–	–	–
LD A,R A ← R			ED 5F	*	*	0	IEF2	0	–
LD ee,mn ee ← mn	ss		01–31	–	–	–	–	–	–
	IX/Y		DD/FD 21						
LD ee,(mn) ee ← (mn)	HL		2A	–	–	–	–	–	–
	ss		ED 4B–7B						
	IX/Y		DD/FD 2A						
LD I,A I ← A			ED 47	–	–	–	–	–	–
LD MB,A † if ADL, MB ← A			ED 6D	–	–	–	–	–	–
LD q,n q ← n	r		06–3E	–	–	–	–	–	–
	ir		DD/FD 26/2E						
	(HL)		36						
	(IX/Y ± d)		DD/FD 36						
LD q,r q ← r	r′		40–7F	–	–	–	–	–	–
	ir		DD/FD 60–6F						
	(HL)		70–77						
	(IX/Y ± d)		DD/FD 70–77						
LD qq,(HL) † qq ← (HL)	BC,DE,HL		ED 07–27	–	–	–	–	–	–
	IX		ED 37						
	IY		ED 31						
LD qq,(IX/Y ± d) † qq ← (IX/Y ± d)	BC,DE,HL		DD/FD 07–27	–	–	–	–	–	–
	same I		DD/FD 37						
	other I		DD/FD 31						
LD R,A R ← A			ED 4F	–	–	–	–	–	–

TABLE 19. INSTRUCTION SUMMARY (CONTINUED)

	Address Mode		Op Code(s) (Hex)	Flags Affected					
Instruction and Operation	**Dest**	**Source**		**S**	**Z**	**HC**	**P/V**	**N**	**CF**
LD r,s r ← s		r′	40–7F	–	–	–	–	–	–
	rae	ih	DD/FD 44–7C (note 1)						
	rae	il	DD/FD 45–7D (note 1)						
		n	06–3E						
		(HL)	46–7E						
		(IX/Y ± d)	DD/FD 46–7E						
LD SP,rr SP ← rr		HL	F9	–	–	–	–	–	–
		IX/Y	DD/FD F9						
LDD (DE) ← (HL) DE ← DE – 1 HL ← HL – 1 BC ← BC – 1			ED A8	–	–	0	NZ	0	–
LDDR do {(DE) ← (HL) DE ← DE – 1 HL ← HL – 1 BC ← BC – 1 } while BC != 0			ED B8	–	–	0	0	0	–
LDI (DE) ← (HL) DE ← DE + 1 HL ← HL + 1 BC ← BC – 1			ED A0	–	–	0	NZ	0	–
LDIR do {(DE) ← (HL) DE ← DE + 1 HL ← HL + 1 BC ← BC – 1 } while BC != 0			ED B0	–	–	0	0	0	–
LEA qq,IX ± d † qq ← IX ± d	BC,DE,HL		ED 02–22	–	–	–	–	–	–
	IX		ED 32						
	IY		ED 55						
LEA qq,IY ± d † qq ← IY ± d	BC,DE,HL		ED 03–23	–	–	–	–	–	–
	IX		ED 54						
	IY		ED 33						
MLT ss ss ← ss_L * ss_H			ED 4C–7C	–	–	–	–	–	–

TABLE 19. INSTRUCTION SUMMARY (CONTINUED)

	Address Mode		Op Code(s) (Hex)	Flags Affected					
Instruction and Operation	**Dest**	**Source**		**S**	**Z**	**HC**	**P/V**	**N**	**CF**
NEG A ← 0 – A			ED 44	*	*	*	V	1	*
NOP			00	–	–	–	–	–	–
OR A,s A ← A OR s		r	B0–B7	*	*	0	P	0	0
		ir	DD/FD B4–B5						
		n	F6						
		(HL)	B6						
		(IX/Y ± d)	DD/FD B6						
OTD2R † do {(BC) ← (HL) B ← B – 1 C ← C – 1 HL ← HL – 1 } while B != 0			ED BC	X	1	X	X	1	–
OTDM (0,C) ← (HL) B ← B – 1 C ← C – 1 HL ← HL – 1			ED 8B	*	*	*	P	*	*
OTDMR do {(0,C) ← (HL) B ← B – 1 C ← C – 1 HL ← HL – 1 } while B != 0			ED 9B	0	1	0	1	*	0
OTDR do {(BC) ← (HL) B ← B – 1 HL ← HL – 1 } while B != 0			ED BB	X	1	X	X	1	–
OTI2R † do {(BC) ← (HL) B ← B – 1 C ← C + 1 HL ← HL + 1 } while B != 0			ED B4	X	1	X	X	1	–
OTIM (0,C) ← (HL) B ← B – 1 C ← C + 1 HL ← HL + 1			ED 83	*	*	*	P	*	*

TABLE 19. INSTRUCTION SUMMARY (CONTINUED)

	Address Mode		Op Code(s) (Hex)	Flags Affected					
Instruction and Operation	**Dest**	**Source**		**S**	**Z**	**HC**	**P/V**	**N**	**CF**
OTIMR do {(0,C) ← (HL) B ← B – 1 C ← C + 1 HL ← HL + 1 } while B != 0			ED 93	0	1	0	1	*	0
OTIR do {(BC) ← (HL) B ← B – 1 HL ← HL + 1 } while B != 0			ED B3	X	1	X	X	1	–
OUT (C),r (BC) ← r			ED 41–79	–	–	–	–	–	–
OUT (n),A (n) ← A			D3	–	–	–	–	–	–
OUT0 (n),r (0,n) ← r			ED 01–39	–	–	–	–	–	–
OUTD (BC) ← (HL) B ← B – 1 HL ← HL – 1			ED AB	X	*	X	X	1	–
OUTD2 † (BC) ← (HL) B ← B – 1 C ← C – 1 HL ← HL – 1			ED AC	X	*	X	X	1	–
OUTI (BC) ← (HL) B ← B – 1 HL ← HL + 1			ED A3	X	*	X	X	1	–
OUTI2 † (BC) ← (HL) B ← B – 1 C ← C + 1 HL ← HL + 1			ED A4	X	*	X	X	1	–
PEA IX/Y ± d † SP ← SP–2 (SP) ← IX/Y ± d			ED 65/66	–	–	–	–	–	–
POP pp pp ← (SP) SP ← SP + 2	qq IX/Y		C1–F1 DD/FD E1	(no change unless operand is AF)					
PUSH pp SP ← SP–2 (SP) ← pp		qq IX/Y	C5–F5 DD/FD E5	–	–	–	–	–	–

TABLE 19. INSTRUCTION SUMMARY (CONTINUED)

	Address Mode		Op Code(s) (Hex)	Flags Affected					
Instruction and Operation	**Dest**	**Source**		**S**	**Z**	**HC**	**P/V**	**N**	**CF**
RES b,m m ← m and not (2^b)	r		CB 80–BF	–	–	–	–	–	–
	(HL)		CB 86–BE						
	(IX/Y ± d)		DD/FD CB d 86–BE						

RET †
if .16 OR .24 {
newADL ← (SPL)
SPL ← SPL + 1
if !ADL {
if newADL {
PC23-16 ← (SPL)
SPL ← SPL + 1}
PC15-0 ← (SPS)
SPS ← SPS + 2}
else [ADL is 1] {
if newADL {
PC23-0 ← (SPL)
SPL ← SPL + 3
} else {
PC15-0 ← (SPL)
SPL ← SPL + 2}}
ADL ← newADL}
else [no prefix] {
if ADL {
PC23-0 ← (SPL)
SPL ← SPL + 3
} else {
PC15-0 ← (SPS)
SPS ← SPS + 2}}
			C9	–	–	–	–	–	–

RET cc if cc {as RET above †}			C0–F8	–	–	–	–	–	–
RETI as RET above † recognition by Z80 peripherals			ED 4D	–	–	–	–	–	–
RETN as RET above † IEF1 ← IEF2			ED 45	–	–	–	–	–	–
RL m (CF,m) ← rotL(CF,m)	r		CB 10–17	*	*	0	P	0	*
	(HL)		CB 16						
	(IX/Y ± d)		DD/FD CB d 16						
RLA (CF,A) ← rotL(CF,A)			17	–	–	0	–	0	*

TABLE 19. INSTRUCTION SUMMARY (CONTINUED)

Instruction and Operation	Address Mode Dest	Address Mode Source	Op Code(s) (Hex)	S	Z	HC	P/V	N	CF
RLC m	r		CB 00–07	*	*	0	P	0	*
C ← 7 0 ←	(HL)		CB 06						
(CF,m) ← rotL(m)	(IX/Y ± d)		DD/FD CB d 06						
RLCA C ← 7 0 ← (CF,A) ← rotL(A)			07	–	–	0	–	0	*
RLD tmp ← A[3:0] A[3:0] ← (HL)[7:4] (HL)[7:4] ← (HL)[3:0] (HL)[3:0] ← tmp			ED 6F	*	*	0	P	0	–
RR m	r		CB 18–1F	*	*	0	P	0	*
→ C → 7 0	(HL)		CB 1E						
(CF,m) ← rotR(CF,m)	(IX/Y ± d)		DD/FD CB d 1E						
RRA → C → 7 0 (CF,A) ← rotR(CF,A)	r		1F	–	–	0	–	0	*
RRC m	r		CB 08–0F	*	*	0	P	0	*
→ C → 7 0	(HL)		CB 0E						
(CF,m) ← rotR(m)	(IX/Y ± d)		DD/FD CB d 0E						
RRCA → C → 7 0 (CF,A) ← rotR(A)			0F	–*	–	0	–	0	*
RRD tmp ← (HL)[3:0] (HL)[3:0] ← (HL)[7:4] (HL)[7:4] ← A[3:0] A[3:0] ← tmp			ED 67	*	*	0	P	0	–
RSMIX † mix_flag ← 0			ED 7E	–	–	–	–	–	–

TABLE 19. INSTRUCTION SUMMARY (CONTINUED)

Instruction and Operation	Address Mode Dest	Address Mode Source	Op Code(s) (Hex)	S	Z	HC	P/V	N	CF
RST p SP ← SP – 2 (SP) ← PC IF ADL {SPL ← SPL – 1 (SPL) ← PC23-0} IF .i16 OR .i24 { SPL ← SPL-1 (SPL) ← ADL ADL ← .i16 ? 0 : 1} PC ← 0,p note p = 0,8,10,18,...38H			C7–FF	*	*	0	P	0	–
SBC A,s A ← A – s – CF		r	98–9F	*	*	*	V	1	*
		ir	DD/FD 9C–9D						
		n	DE						
		(HL)	9E						
		(IX/Y ± d)	DD/FD 9E						
SBC HL,ss HL ← HL – ss – CF		r	ED 42–72	*	*	*	V	1	*
SCF CF ← 1			37	–	–	0	–	0	1
SET b,m m ← m or (2^b)	r		CB C0–FF	–	–	–	–	–	–
	(HL)		CB C6–FE						
	(IX/Y ± d)		DD/FD CB d C6–FE						
SLA m C ← 7 0 ← 0 (CF,m) ← m + m	r		CB 20–27	*	*	0	P	0	*
	(HL)		CB 26						
	(IX/Y ± d)		DD/FD CB d 26						
SLP			ED 76	–	–	–	–	–	–
SRA m 7 0 → C (m,CF) ← arith_shR(m)	r		CB 28–2F	*	*	0	P	0	*
	(HL)		CB 2E						
	(IX/Y ± d)		DD/FD CB d 2E						
SRL m 0 → 7 0 → C (m,CF) ← logic_shR(m)	r		CB 38–3F	0	*	0	P	0	*
	(HL)		CB 3E						
	(IX/Y ± d)		DD/FD CB d 3E						
STMIX † mix_flag ← 1			ED 7D	–	–	–	–	–	–

TABLE 19. INSTRUCTION SUMMARY (CONTINUED)

	Address Mode		Op Code(s) (Hex)	Flags Affected					
Instruction and Operation	**Dest**	**Source**		**S**	**Z**	**HC**	**P/V**	**N**	**CF**
SUB A,s A ← A – s		r	90–97	*	*	*	V	1	*
		ir	DD/FD 94–95						
		n	D6						
		(HL)	96						
		(IX/Y ± d)	DD/FD 96						
TST A,s A AND s		r	ED 04–3C	*	*	1	P	0	0
		n	ED 64						
		(HL)	ED 34						
TSTIO n (0,C) AND n			ED 34	*	*	1	P	0	0
XOR A,s A ← A XOR s		r	A8–AF	*	*	0	P	0	0
		ir	DD/FD AC–AD						
		n	EE						
		(HL)	AE						
		(IX/Y ± d)	DD/FD AE						

NOTE: Some of the values in this range are used by other instructions, which override this range.

OP CODE MAP

TABLE 20. OP CODE MAP (FIRST OP CODE)

LOWER NIBBLE (HEX) across; UPPER NIBBLE (HEX) down

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	LD BC,nn	LD (BC),A	INC BC	INC B	DEC B	LD B,n	RLCA	EX AF,AF'	ADD HL,BC	LD A,(BC)	DEC BC	INC C	DEC C	LD C,n	RRCA
1	DJNZ d	LD DE,nn	LD (DE),A	INC DE	INC D	DEC D	LD D,n	RLA	JR d	ADD HL,DE	LD A,(DE)	DEC DE	INC E	DEC E	LD E,n	RRA
2	JR NZ,d	LD HL,nn	LD (nn),HL	INC HL	INC H	DEC H	LD H,n	DAA	JR Z,d	ADD HL,HL	LD (HL),nn	DEC HL	INC L	DEC L	LD L,n	CPL
3	JR NC,d	LD SP,nn	LD (nn),A	INC SP	INC (HL)	DEC (HL)	LD (HL),n	SCF	JR C,d	ADD HL,SP	LD A,(nn)	DEC SP	INC A	DEC A	LD A,n	CCF
4	.16.i16 prefix	LD B,C	LD B,D	LD B,E	LD B,H	LD B,L	LD B,(HL)	LD C,A	LD C,B	.24.i16 prefix	LD C,D	LD C,E	LD C,H	LD C,L	LD C,(HL)	LD C,A
5	LD D,B	LD D,C	.16.i24 prefix	LD D,E	LD D,H	LD D,L	LD D,(HL)	LD D,A	LD E,B	LD E,C	LD E,D	.24.i24 prefix	LD E,H	LD E,L	LD E,(HL)	LD E,A
6	LD H,B	LD H,C	LD H,D	LD H,E	LD H,H	LD H,L	LD H,(HL)	LD H,A	LD L,B	LD L,C	LD L,D	LD L,E	LD L,H	LD L,L	LD L,(HL)	LD L,A
7	LD (HL),B	LD (HL),C	LD (HL),D	LD (HL),E	LD (HL),H	LD (HL),L	HALT	LD (HL),A	LD A,B	LD A,C	LD A,D	LD A,E	LD A,H	LD A,L	LD A,(HL)	LD A,A
8	ADD A,B	ADD A,C	ADD A,D	ADD A,E	ADD A,H	ADD A,L	ADD A,(HL)	ADC A,A	ADC A,B	ADC A,C	ADC A,D	ADC A,E	ADC A,H	ADC A,L	ADC A,(HL)	ADC A,A
9	SUB A,B	SUB A,C	SUB A,D	SUB A,E	SUB A,H	SUB A,L	SUB A,(HL)	SUB A,A	SBC A,B	SBC A,C	SBC A,D	SBC A,E	SBC A,H	SBC A,L	SBC A,(HL)	SBC A,A
A	AND A,B	AND A,C	AND A,D	AND A,E	AND A,H	AND A,L	AND A,(HL)	AND A,A	XOR A,B	XOR A,C	XOR A,D	XOR A,E	XOR A,H	XOR A,L	XOR A,(HL)	XOR A,A
B	OR A,B	OR A,C	OR A,D	OR A,E	OR A,H	OR A,L	OR A,(HL)	OR A,A	CP A,B	CP A,C	CP A,D	CP A,E	CP A,H	CP A,L	CP A,(HL)	CP A,A
C	RET NZ	POP BC	JP NZ,nn	JP nn	CALL NZ,nn	PUSH BC	ADD A,n	RST 0	RET Z	RET	JP Z,nn	(Table 21)	CALL Z,nn	CALL nn	ADC A,n	RST 8
D	RET NZ	POP DE	JP NC,nn	OUT (n),A	CALL NC,nn	PUSH DE	SUB A,n	RST 10H	RET C	EXX	JP C,nn	IN A,(n)	CALL C,nn	(Table 22)	SBC A,n	RST 18H
E	RET PO	POP HL	JP PO,nn	EX (SP),HL	CALL PO,nn	PUSH HL	AND A,n	RST 20	RET PE	JP (HL)	JP PE,nn	EX DE,HL	CALL PE,nn	(Table 23)	XOR A,n	RST 28H
F	RET P	POP AF	JP P,nn	DI	CALL P,nn	PUSH AF	OR A,n	RST 30H	RET M	LD SP,HL	JP M,nn	EI	CALL M,nn	(Table 24)	CP A,n	RST 38H

Notes:
n = 8-bit data
nn = 16-bit addr or data
d = signed 8-bit displacement

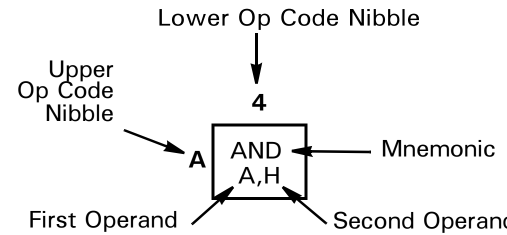

TABLE 21. OP CODE MAP (SECOND OP CODE AFTER 0CBH)

LOWER NIBBLE (HEX)

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

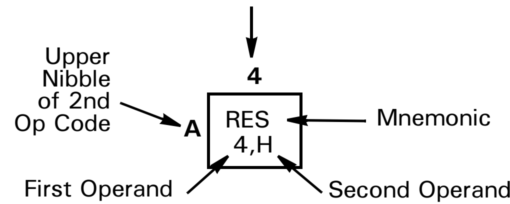

TABLE 22. OP CODE MAP (SECOND OP CODE AFTER 0DDH)

LOWER NIBBLE (HEX) across the top; UPPER NIBBLE (HEX) down the side

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0								LD BC, (IX ± d)		ADD IX,BC						LD (IX ± d),BC
1								LD DE, (IX ± d)		ADD IX,DE						LD (IX ± d),DE
2		LD IX,nn	LD (nn),IX	INC IX	INC IXH	DEC IXH	LD IXH,n	LD HL, (IX ± d)		ADD IX,IX	LD IX,(nn)	DEC IX	INC IXL	DEC IXL	LD IXL,n	LD (IX ± d),HL
3		LD IY, (IX ± d)			INC (IX ± d)	DEC (IX ± d)	LD (IX ± d),n	LD IX, (IX ± d)		ADD IX,SP					LD (IX ± d),IY	LD (IX ± d),IX
4					LD B,IXH	LD B,IXL	LD B, (IX ± d)						LD C,IXH	LD C,IXL	LD C, (IX ± d)	
5					LD D,IXH	LD D,IXL	LD D, (IX ± d)						LD E,IXH	LD E,IXL	LD E, (IX ± d)	
6	LD IXH,B	LD IXH,C	LD IXH,D	LD IXH,E	LD IXH,H	LD IXH,L	LD H, (IX ± d)	LD IXH,A	LD IXL,B	LD IXL,C	LD IXL,D	LD IXL,E	LD IXL,H	LD IXL,L	LD L, (IX ± d)	LD IXL,A
7	LD (IX ± d),B	LD (IX ± d),C	LD (IX ± d),D	LD (IX ± d),E	LD (IX ± d),H	LD (IX ± d),L		LD (IX ± d),A					LD A,IXH	LD A,IXL	LD A, (IX ± d)	
8					ADD A,IXH	ADD A,IXL	ADD A, (IX ± d)						ADC A,IXH	ADC A,IXL	ADC A, (IX ± d)	
9					SUB A,IXH	SUB A,IXL	SUB A, (IX ± d)						SBC A,IXH	SBC A,IXL	SBC A, (IX ± d)	
A					AND A,IXH	AND A,IXL	AND A, (IX ± d)						XOR A,IXH	XOR A,IXL	XOR A, (IX ± d)	
B					OR A,IXH	OR A,IXL	OR A, (IX ± d)						CP A,IXH	CP A,IXL	CP A, (IX ± d)	
C												(Table 25)				
D																
E		POP IX		EX (SP),IX		PUSH IX				JP (IX)						
F										LD SP,IX						
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**

Notes:
n = 8-bit data
nn = 16-bit addr or data
d = signed 8-bit displacement

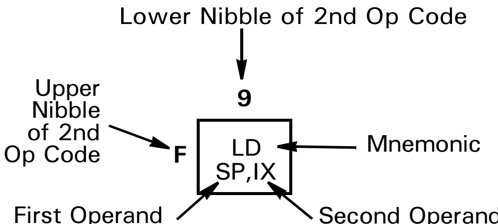

TABLE 23. OP CODE MAP SECOND OP CODE AFTER 0EDH)

LOWER NIBBLE (HEX) across; UPPER NIBBLE (HEX) down

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	IN0 B,(n)	OUT0 (n),B	LEA BC ,IX±d	LEA BC ,IY±d	TST A,B			LD BC, (HL)	IN0 C,(n)	OUT0 (n),C			TST A,C			LD (HL) ,BC
1	IN0 D,(n)	OUT0 (n),D	LEA DE ,IX±d	LEA DE ,IY±d	TST A,D			LD DE, (HL)	IN0 E,(n)	OUT0 (n),E			TST A,E			LD (HL) ,DE
2	IN0 H,(n)	OUT0 (n),H	LEA HL ,IX±d	LEA HL ,IY±d	TST A,H			LD HL, (HL)	IN0 L,(n)	OUT0 (n),L			TST A,L			LD (HL) ,HL
3	IN0 F,(n)	LD IY, (HL)	LEA IX ,IX±d	LEA IY ,IY±d	TST A,(HL)			LD IX, (HL)	IN0 A,(n)	OUT0 (n),A			TST A,A		LD (HL) ,IY	LD (HL) ,IX
4	IN B,(C)	OUT (C),B	SBC HL,BC	LD (nn),BC	NEG	RETN	IM 0	LD I,A	IN C,(C)	OUT (C),C	ADC HL,BC	LD BC,(nn)	MLT BC	RETI		LD R,A
5	IN D,(C)	OUT (C),D	SBC HL,DE	LD (nn),DE	LEA IX ,IY±d	LEA IY ,IX±d	IM 1	LD A,I	IN E,(C)	OUT (C),E	ADC HL,DE	LD DE,(nn)	MLT DE		IM 2	LD A,R
6	IN H,(C)	OUT (C),H	SBC HL,HL	LD (nn),HL	TST A,n	PEA IX±d	PEA IY±d	RRD	IN L,(C)	OUT (C),L	ADC HL,HL	LD HL,(nn)	MLT HL	LD MB,A	LD A,MB	RLD
7	IN F,(C)		SBC HL,SP	LD (nn),SP	TSTIO n		SLP		IN A,(C)	OUT (C),A	ADC HL,SP	LD SP,(nn)	MLT SP	STMIX	RSMIX	
8			INIM	OTIM	INI2						INDM	OTDM	IND2			
9			INIMR	OTIMR	INI2R						INDMR	OTDMR	IND2R			
A	LDI	CPI	INI	OUTI	OUTI2				LDD	CPD	IND	OUTD	OUTD2			
B	LDIR	CPIR	INIR	OTIR	OTI2R				LDDR	CPDR	INDR	OTDR	OTD2R			
C																
D																
E																
F																

n = 8-bit data
nn = 16-bit addr or data
d = signed 8-bit displacement

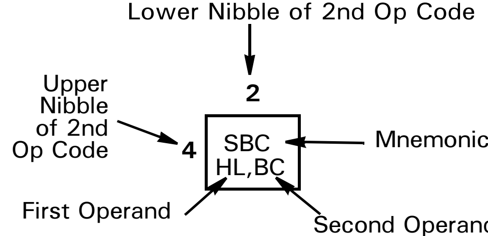

TABLE 24. OP CODE MAP SECOND OP CODE AFTER 0FDH)

LOWER NIBBLE (HEX) across the top; UPPER NIBBLE (HEX) down the side.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0								LD BC, (IY ± d)		ADD IY,BC						LD (IY ± d),BC
1								LD DE, (IY ± d)		ADD IY,DE						LD (IY ± d),DE
2		LD IY,nn	LD (nn),IY	INC IY	INC IYH	DEC IYH	LD IYH,n	LD HL, (IY ± d)		ADD IY,IY	LD IY,(nn)	DEC IY	INC IYL	DEC IYL	LD IYL,n	LD (IY ± d),HL
3		LD IX, (IY ± d)			INC (IY ± d)	DEC (IY ± d)	LD (IY ± d),n	LD IY, (IY ± d)		ADD IY,SP					LD (IY ± d),IX	LD (IY ± d),IY
4					LD B,IYH	LD B,IYL	LD B, (IY ± d)						LD C,IYH	LD C,IYL	LD C, (IY ± d)	
5					LD D,IYH	LD D,IYL	LD D, (IY ± d)						LD E,IYH	LD E,IYL	LD E, (IY ± d)	
6	LD IYH,B	LD IYH,C	LD IYH,D	LD IYH,E	LD IYH,H	LD IYH,L	LD H, (IY ± d)	LD IYH,A	LD IYL,B	LD IYL,C	LD IYL,D	LD IYL,E	LD IYL,H	LD IYL,L	LD L, (IY ± d)	LD IYL,A
7	LD (IY ± d),B	LD (IY ± d),C	LD (IY ± d),D	LD (IY ± d),E	LD (IY ± d),H	LD (IY ± d),L		LD (IY ± d),A					LD A,IYH	LD A,IYL	LD A, (IY ± d)	
8					ADD A,IYH	ADD A,IYL	ADD A, (IY ± d)						ADC A,IYH	ADC A,IYL	ADC A, (IY ± d)	
9					SUB A,IYH	SUB A,IYL	SUB A, (IY ± d)						SBC A,IYH	SBC A,IYL	SBC A, (IY ± d)	
A					AND A,IYH	AND A,IYL	AND A, (IY ± d)						XOR A,IYH	XOR A,IYL	XOR A, (IY ± d)	
B					OR A,IYH	OR A,IYL	OR A, (IY ± d)						CP A,IYH	CP A,IYL	CP A, (IY ± d)	
C												(Table 26)				
D																
E		POP IY		EX (SP),IY		PUSH IY				JP (IY)						
F										LD SP,IY						

Notes:
n = 8-bit data
nn = 16-bit addr or data
d = signed 8-bit displacement

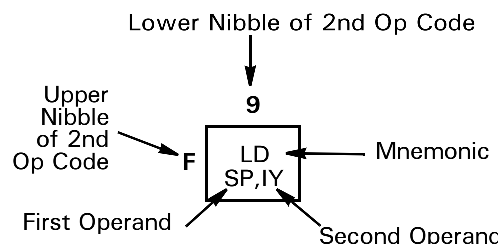

TABLE 25. OP CODE MAP (4TH BYTE, AFTER 0DDH, 0CBH, AND d)

LOWER NIBBLE (HEX) across the top; UPPER NIBBLE (HEX) down the side.

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

Notes:
d = signed 8-bit displacement

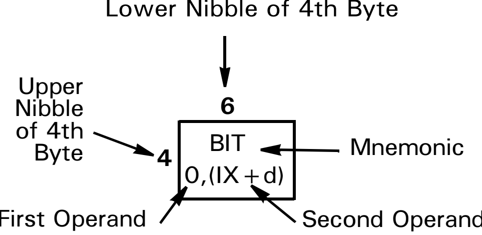

TABLE 26. OP CODE MAP (4TH BYTE, AFTER 0FDH, 0CBH, AND d)

LOWER NIBBLE (HEX)

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

Notes:
d = signed 8-bit displacement