eZ80

GENERAL 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.

Considering both the increased clock speed and processor efficiency, the eZ80's processing power rivals the performance of 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, or full 24-bit (16 MB) address mode.

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

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

ARCHITECTURAL OVERVIEW

The eZ80 is ZiLOG's next-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

PIN DESCRIPTIONS

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

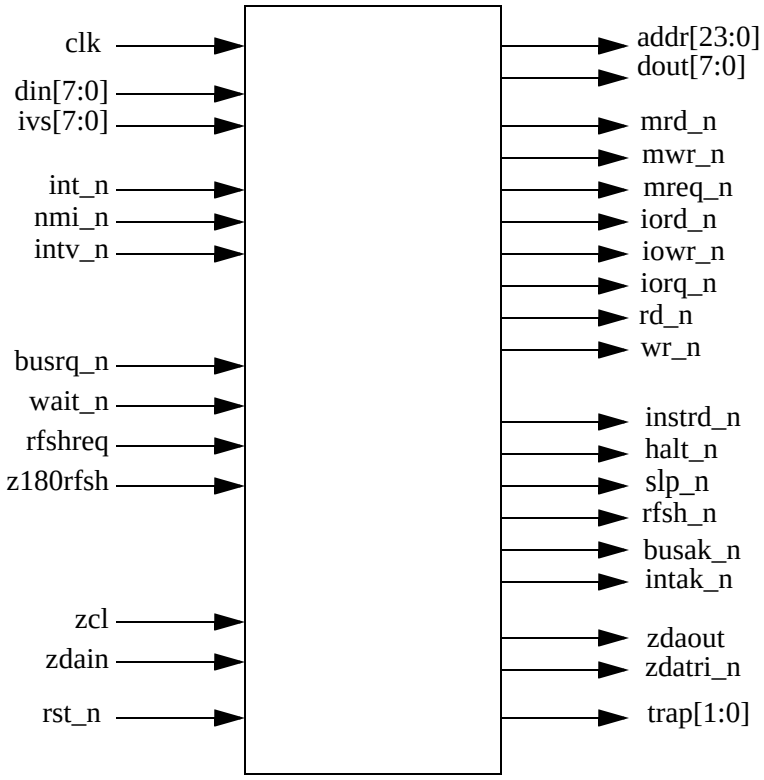

FIGURE 1. EZ80 LOGIC DIAGRAM

TABLE 1. PROCESSOR AND DEVICE PIN DESCRIPTIONS

Symbol	Function	Type	Description
addr[23–0	Address Bus	Output	These lines select a location in memory or I/O space to be read or written.
busak_n	Bus Acknowledge	Output, active Low	The eZ80 responds to a Low on busrq_n, by suspending instruction execution and driving this line Low.
busrq_n	Bus request	Input, active Low	External devices can force the eZ80 to suspend operation driving this line Low.
clk	Clock	Input	The master clock of the eZ80.
din[7:0]	Data Bus In	Input	These lines transfer information from I/O and memory devices to the eZ80.
dout[7:0]	Data Bus Out	Output	These lines transfer information from the eZ80 to I/O and memory devices.
halt_n	Halt	Output, active Low	A low on this pin indicates that the eZ80 is stopped because of a HALT instruction.
instrd_n	Instruction Read	Output, active Low	When instrd_n is Low, it indicates that the eZ80 is fetching an instruction from memoryeZ80.
int_n	Interrupt Input	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.
intak_n	Interrupt Acknowledge	Output, active Low	A Low indicates that the eZ80 is acknowledging an interrupt request on int_n.
intv_v	Vectored Interrupt Input	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.
iord_n	I/O Read	Output, active Low	iord_n 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 din[7:0] bus.
iorq_n	I/O Request	Output, active Low	iorq_n Low indicates that the eZ80 is accessing a location in I/O space. The $\overline{RD}$ and $\overline{WR}$ pins indicate the type of access.
iowr_n	I/O Write	Output, active Low	iowr_n Low indicates that dout[7:0] hold data to be stored at the addressed I/O location.
ivs[7:0]	Interrupt Vector	Input	In response to an interrupt caused by intv_n the low vector byte is latched from this bus by the eZ80.
mrd_n	Memory Read	Output, active Low	A 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 din[7:0] bus.
mreq_n	Memory Request	Output, active Low	A Low indicates that the eZ80 is accessing a location in memory. The rd_n, wr_n, and instrd_n pins indicate the type of access.
mwr_n	Memory Write	Output, active Low	A low indicates that dout[7:0] hold the data to be stored at the addressed memory location.

TABLE 1. PROCESSOR AND DEVICE PIN DESCRIPTIONS (CONTINUED)

Symbol	Function	Type	Description
nmi_n	Nonmaskable Interrupt	Input, falling-edge active	nmi_n has a higher priority than int_n and intv_n 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.
rd_n	Read	Output, active Low	A Low indicates that the eZ80 is reading data from a location in memory or I/O space. The addressed memory or I/O uses this signal to gate data onto the din[7:0] bus.
rfsh_n	Refresh	Ourput, active Low	
rfshreq_n	Refresh request	Input, active Low	
rst_n	Master Reset	Input, active Low	This signal is used to initialize the eZ80.
slp_n	Sleep	Output, active Low	A low on this pin indicates that the eZ80 is stopped because of a SLP instruction.
trap[1:0]	Instruction Trap	Output, active High	A High on either of these pins indicates that the eZ80 has detected an invalid instruction.
wait_n	Wait	Input, active Low	External devices can extend bus cycles to more than one clock, by driving this line low.
wr_n	Write	Output, active Low	A low indicates that dout[7:0] holds the data to be stored at the addressed memory or I/O location.
z180rfsh	Refresh control	Input, active high	
zcl	ZiLOG Debug Clock	Input	Serial clock source for the ZiLOG Debug Interface (ZDI) tool.
zdain	Debug data in	Input	ZDI serial data input.
zdaout	Debug data out	Output	ZDI serial data output
zdatri_n	Debug pin control	Output, active Low	ZDI serial data tri-state control, when Low the package pad should be tri-stated.

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 multi-byte register HL 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 are part of multi-byte registers BC, DE, and HL. Flag register F completes the basic register bank.

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.

Operating Modes

The eZ80 has two addressing modes, 16-bit mode and 24-bit mode. These modes are controled by the Address and Data Long (ADL) bit.

When ADL is 0:

- the PC, SP, BC, DE, HL, IX, and IY registers are effectively 16 bits wide, as on the Z80 and Z80180, and

When ADL is 1:

- the PC, SP, BC, DE, HL, IX, and IY registers are 24 bits wide.

The multiple operating modes of the processor allows Z80 code to be run without change in *native Z80* or *virtual Z80* with ADL cleared to zero or with ADL set to one the application can take advantage of the eZ80's 16-Mbyte linear addressing space and enhanced instruction set.

These operating modes are governed by:

- The Address and Data Long (ADL) mode bit, and
- An 8-bit register called MBASE.

Native Z80 Mode. ADL, and MBASE reset to 0. 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. The upper 8-bits of address (23-16) are held at zero, the value of MBASE. This is the mode the eZ80 comes up in after reset.

Virtual Z80 Mode. If ADL is cleared, 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 in the 16-Mbyte memory space by MBASE. In this Virtual Z80 mode, several tasks can each have their own Z80 partition.

ADL Mode. If ADL is set, MBASE has no 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. .

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.

Nonmaskable Interrupt (NMI). The eZ80 latches falling edges on the nmi_n pin. Only a Low on RESET (rst_n) or BUSRQ (busrq_n) takes precedence over NMI. Unless RESET or BUSRQ is Low, the eZ80 checks for a latched edge from NMI as it completes each instruction and performs an NMI sequence if a falling edge has occurred.

The nonmaskable interrupt always vectors to location 66H for the start of its interrupt routine.

Interrupt (INT). The eZ80 can handle interrupts requested by a device on the int_n pin, in any of three ways called modes 0, 1, or 2. The interrupt from the int_n pin is maskable via the EI and DI instructions which enable or disable interrupts.

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

Interrupt Processor Response. The eZ80 performs an int_n interrupt sequence at the end of an instruction if all of the following are true:

- INT (int_n) is Low
- RESET and BUSRQ are both High
- A negative edge on $\overline{\text{NMI}}$ has not been detected
- Interrupts are enabled

When all of these conditions occur simultaneously, the eZ80 responds with an interrupt acknopwledge cycle.

While all interrupt acknowledge cycles follow a general pattern, they differ as to what the processor does with the data on din[7:0]. These actions depend upon the most recently executed IM instruction.

All interrupt acknowledge cycles:

- Save the address of the interrupted instruction (return address)
- Clear the interrupt enable flag (disables interrupts), preventing further interrupts
- Drives the intak_n pin Low

INT Mode 0. If the most recently executed IM instruction was IM 1, the eZ80 reads the data on din[7:0] as an instruction opcode (while intak_n is Low). If the instruction is a multi-byte instruction (CALL) the intak_n pin will be brought back High and then low again for each sussive instruction opcode bytes.

INT Mode 1. If the most recently executed IM instruction was IM 1, the eZ80 ignors the data on din[7:0] and vectors to address 38H.

INT Mode 2. If the most recently executed IM instruction was IM 2, the eZ80 performs a vectored interrupt. The lower 8-bits of the interrupt vector is placed on din[7:0] while intak_n is Low (bit 0 is assumed to be zero). The actual vector address is made up of bits 23-16 are zero, bits 15-8 are the contenst of the I register, and bits 7-0 are supplied vector byte. The 16-bit word at the above vector address is fetched and its value is assumed to be the start of the interrupt service routine, with bits 23-16 provided by the eZ80 processor as zeros.

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.

Vectored Interrupt. The eZ80 has a vectored interrupt source intv_n which acts simulator to the mode 2 interrupts above. This interrupt can be masked via the EI and DI instructions.

The intv_n interrupt response differs from the INT response in that the vector Low byte is not latched from the data bus but comes from the ivs[7:0] bus.

Ilegal Instruction Traps

The eZ80 instruction set does not fully cover all possible sequences of binary values. Sequences for which no operation is defined, are illegal instructions.

When an eZ80 processor fetches one of these sequences, it performs a Trap sequence. Which byte of the mulit-byte instruction which caused the trap is indicated by the trap[1:0] bus.

Interrupt and Traps. Applications that operate only in Native Z80 mode, or ADL mode, 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, as interrupts and traps are never mapped, applications that switch between modes, or operate in Virtual Z80, mode, can 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. The eZ80 features an I/O space with 16-bit addresses and 64K bytes.

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. The Memory Base (MBASE) register is zero.

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.

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 two modes, or to use MBASE addressing in ADL mode.

Addressing Modes

Memory addresses can be formed 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 start of the instruction, to which program control can be transferred.

Direct Addressing. Direct-addressing instructions include a 16-bit logical or 24-bit linear address, depending on a prefix byte or the ADL mode. (The same operational distinction applies to instructions that contain 16- or 24-bit immediate data.)

Register Indirect Addressing. The address is taken from one of the multi-byte registers `BC`, `DE` or `HL`. Depending on a prefix byte or the ADL mode, the register supplies a 24-bit linear address or a 16-bit logical address that is subject to MBASE.

Indexed Addressing. In this mode, instructions include an 8-bit signed displacement from the address in an index register, IX or IY. Depending on a prefix byte or the ADL mode, the register supplies a 24-bit linear address or a 16-bit logical address that is subject to MBASE.

Instruction Fetching. In ADL mode, the Program Counter (PC) supplies a 24-bit linear address. In other modes, it supplies a 16-bit address that is subject to MBASE.

Stack Operations. Depending on the ADL mode and in some cases on a prefix byte, stack addresses may be formed in either of two ways. Either a 24-bit register SPL acts as the Stack Pointer, and supplies a 24-bit linear address, or the 16-bit register SPS acts as the Stack Pointer, supplying a 16-bit logical address that is subject to MBASE.

Interrupts, Traps, and RST Instructions. All of these operations are affected by a global state called Mixed ADL, which should be set appropriately for each application. Mixed ADL should be 0 for applications in which all code runs in the same ADL state, but should be 1 for applications that include some code that runs in ADL mode and some that runs in other modes. If Mixed ADL is 1, and an interrupt, Trap, or Restart occurs, the eZ80 stacks a byte containing the ADL mode of the interrupted, trapped, or calling process on SPL, before setting ADL mode for the service routine.

Mode 2 Interrupts. in ADL mode, the interrupt table must always be in the first 64K bytes, as must the start of interrupt service routines entered through the interrupt table.

A23-16 for Instruction fetching

If ADL is 1, A23-16 for instruction fetches come from bits 23-16 of the Program Counter, while if ADL is 0 they come from the MBASE register. There is no override facility for this choice.

Indirect register and Indexed addressing

When ADL is 1, A23-16 of the memory address for the execution cycle(s) are taken from the high-order 8 bits of the 24-bit register, while when ADL is 0 these address lines are taken from the MBASE register. These conventions can be overridden for one of these instructions by preceding it with a prefix byte (by suffixing its assembler opcode with ".S" or ".L").

Stack Pointer Selection

The following instructions:

```
EX (SP),HL/IX/IY
PUSH
POP
LD SP,(nnnn)
LD (nnnn),SP
LD SP,nnnn
```

use the 24-bit SPL register if ADL is 1, while if ADL is 0 they use the 16-bit SPS register, mapped by the MBASE. This convention can be overridden for one instruction by preceding it with a prefix byte (assembler opcode suffix ".S" or ".L").

Direct Addresses

While fetching the instructions:

```
LD r,(nnnn)
LD (nnnn),r
LD rr,(nnnn)
LD (nnnn),rr,
```

the processor fetches three address bytes if ADL is 1, while if ADL is 0 it fetches only two bytes of address, and uses the MBASE for A23-16. This convention can be overridden for one instruction by preceding it with a prefix (assembler opcode suffix .IS or .IL).

Multi-byte immediate data

While fetching the instruction

```
LD rr,nnnn
```

the processor fetches three data bytes if ADL is 1, while if ADL is 0 it fetches only two data bytes, and clears bits 23-16 of the register to zero. This convention can be overridden for one instruction by preceding it with a prefix (assembler opcode suffix .IS or .IL).

16 vs. 24-bit memory data

In execution/stack cycles for:

```
EX (SP),HL/IX/IY
LD BC/DE/HL/SP/IX/IY,(nnnn)
LD (nnnn),BC/DE/HL/SP/IX/IY
LD BC/DE/HL/IX/IY,(HL/IX+d/IY+d)
LD (HL/IX+d/IY+d),BC/DE/HL/IX/IY
PUSH*
POP*
```

the processor stores and/or fetches three bytes in memory if ADL is 1, or two bytes if ADL is 0. For memory-fetch cycles (including POP) when ADL is zero, the processor zeroes bits 23-16 of the affected register. This convention can be overridden for one instruction by preceding it with a prefix (assembler opcode suffix .S or .L).

* PUSH AF and POP AF with ADL=1 are special cases. For these, SP is incremented or decremented by 3, as for other stack operations when ADL is 1, but the processor only reads or writes two bytes.

Internal 16- vs. 24-bit operations

In instructions:

```
LD SP,HL/IX/IY
EX DE,HL
LDI, LDIR, LDD, LDDR
CPI, CPIR, CPD, CPDR
ADD/SUB/SBC HL,rr
ADD IX/IY,rr
INC/DEC rr
JR e
JR cc,e
DJNZ e
INI, INIR, INI2, INI2R, IND, INDR
OUTI, OTIR, OTI2, OTI2R, OUTD, OTDR
```

if ADL is 1 the operation affects all 24 bits of registers that are loaded or modified, while if ADL is 0, bits 23-16 of affected registers are zeroed. This convention can be overridden for one instruction by preceding it with a prefix (assembler opcode suffix .S or .L).

Condition Codes

In instructions

`ADD/ADC/SBC rr,rr`
and BC decrementing in `LDI, LDIR, LDD, LDDR, CPI, CPIR, CPD, CPDR`

the result condition code reflects the 24-bit result if ADL is 1, else it reflects the 16-bit result as on the 18x. This convention can be overridden for one instruction by preceding it with a prefix (assembler opcode suffix .S or .L).

Changing the ADL Mode: CALL, RST, JP, and RET

There is no separate instruction to simply change ADL, because after such an instruction the Program Counter would undergo an unmanageable change in interpretation. ADL can be changed only by prefixing a CALL or JP nnnn instruction with a .IS or .IL prefix, or a RST, RET or JP (rr) instruction with a .S or .L prefix.

Tables 2 through 6 describe these instructions for various cases of prefix bytes and the ADL mode.

TABLE 2. CALL INSTRUCTION

ADL	Prefix	Operation
0	none	stack 2-byte logical return address using SPS mapped by MBASE keep ADL 0 load a 2 byte logical address from the instruction into PC
1	none	stack the 3 byte return address using SPL keep ADL 1 load a 3 byte address from the instruction into PC
0	.IS	stack 2-byte logical return address using SPS mapped by MBASE stack a 00 byte using SPL keep ADL 0 load a 2 byte logical address from the instruction into PC
1	.IS	stack the 2 LS bytes of the return address using SPS mapped by MBASE stack the MS byte of the return address using SPL stack a 01 byte using SPL clear ADL load a 2 byte logical address from the instruction into PC
0	.IL	stack the 2 byte logical return address using SPL stack a 00 byte using SPL set ADL load a 3 byte address from the instruction into PC
1	.IL	stack the 3 byte return address using SPL stack a 01 byte using SPL keep ADL 1 load a 3 byte address from the instruction into PC

TABLE 3. RST NN INSTRUCTION

ADL	Prefix	Operation
0	none	stack 2-byte logical return address using SPS mapped by MBASE keep ADL 0 load the 16-bit logical address 00nn into PC
1	none	stack the 3 byte return address using SPL keep ADL 1 load the 24-bit address 0000nn into PC
0	.IS	stack 2-byte logical return address using SPS mapped by MBASE stack a 00 byte using SPL keep ADL 0 load the 16-bit logical address 00nn into PC
1	.IS	stack the 2 LS bytes of the return address using SPS mapped by MBASE stack the MS byte of the return address using SPL stack a 01 byte using SPL clear ADL load the 16-bit logical address 00nn into PC.
0	.IL	stack the 2 byte logical return address using SPL stack a 00 byte using SPL set ADL load the 24-bit address 0000nn into PC
1	.IL	stack the 3 byte return address using SPL stack a 01 byte using SPL keep ADL 1 load the 24-bit address 0000nn into PC

TABLE 4. JP NNNN INSTRUCTION

ADL	Prefix	Operation
0	none	load a 2-byte logical address from the instruction into PC keep ADL 0
1	none	load a 3-byte address from the instruction into PC keep ADL 1
x	.IS	clear ADL load a 2-byte logical address from the instruction into PC
x	.IL	set ADL load a 3-byte address from the instruction into PC

TABLE 5. RET, RETI, OR RETN INSTRUCTION

ADL	Prefix	Operation
0	none	pop a 2-byte logical address from SPS mapped by MBASE into PC keep ADL 0
1	none	pop a 3-byte logical address from SPL into PC keep ADL 1

TABLE 5. RET, RETI, OR RETN INSTRUCTION

ADL	Prefix	Operation
0	.S or .L	pop a byte from SPL load its units bit into ADL if ADL is still 0, pop 2-byte logical address from SPS mapped by MBASE into PC if ADL is now 1, pop a byte from SPL into PC23-16, then pop two bytes from SPS mapped by MBASE into PC15-0.
1	.S or .L	pop a byte from SPL load its units bit into ADL if ADL is now 0, pop a 2-byte logical address from SPL into PC if ADL is still 1, pop a 3-byte address from SPL into PC.

TABLE 6. JP (RR) INSTRUCTION

ADL	Prefix	Operation
0	none	load a 16-bit logical address from the register into PC keep ADL 0
1	none	load a 24-bit address from the register into PC keep ADL 1
x	.IS	clear ADL load a 16-bit logical address from the register into PC
x	.IL	set ADL load a 24-bit address from the register into PC

Mixed-ADL Applications

Applications that include legacy routines/functions/tasks/modules that run in non-ADL mode, and new routines/functions/tasks/modules that run in ADL mode, must follow certain rules to assure proper operation.

1. Include a STMIX instruction in device initialization, to assure that interrupt service routines begin in a consistent mode (ADL mode).
2. End all interrupt service routines with a prefixed RET or RETI instruction, which POPs the interrupted code's ADL state from the SPL stack.
3. CALL, or JP to, each routine/function/task/module in the mode in which it was assembled or compiled.
4. Any routine that may be CALLed from either mode, must be CALLed with a prefix to save the caller's ADL mode on the SPL stack.
5. Any routine that may be CALLed from either mode, must return with a prefixed RET instruction, to restore the caller's ADL state from the SPL stack.
6. If calling code operating in one mode must pass stack-based operands/arguments to a routine compiled or assembled for a different mode, it must use prefixed instructions to set up the operands/arguments. For PUSH, .S and .L

prefixes control both whether SPS or SPL is used, and whether the operands/ arguments are stored as 2-byte or 3-byte values.

NOTE: In mixed-ADL applications, some of the rules above may represent exceptions to the eZ80's design goal that legacy code not have to be modified in order to be run on the eZ80. Assuming that legacy routines are not selectively converted to ADL mode, and that legacy routines don't call newly-written routines, the only rule that would lead to such modification would be number 5. If each legacy routine ends with a single RET at its end, this conversion is easy. Internal and conditional RETs require more careful review -- a program to highlight RETs may be helpful.

Prefix Bytes: Exceptions to the ADL Mode

In the ZiLOG ZMASM / ZDS assembler, code is assembled for a given state of the ADL mode bit by preceding it with a pseudo-op:

```
.assume adl=1 ; or 0
```

The programmer is of course responsible for ensuring that this source-file setting matches the state of the hardware ADL mode bit when the code is executed.

The ADL mode and `assume` setting govern several different aspects of instruction operation and eZ80 operation, as described in preceding sections. Two different kinds of exception to normal operation can be selected for a particular instruction, by adding a suffix to the instruction's op code:

Suffices .IS and .IL control whether a memory address or multi-byte immediate data, in the instruction, should be two or three bytes long.

Suffices .S and .L control whether the overall operation of the instruction should involve 16 bits or 24 bits.

Table 7 shows which suffices apply to which instructions. Instructions not shown in this table are not affected by prefix bytes / opcode suffixes.

TABLE 7. APPLICABILITY OF EXCEPTION SUFFIXES

Instruction (Class)	.S/.L	.IS/.IL
`ADD/ADC/SUB/SBC/AND/OR/XOR/CP A,(HL/IX+d/IY+d)`	Y	
`ADD/ADC/SBC rr,rr`	Y	
`BIT/SET/RES b,(HL/IX+d/IY+d)`	Y	
`CALL`		Y
`CPI, CPIR, CPD, CPDR`	Y	
`EX DE,HL`	Y	
`EX (SP),HL/IX/IY`	Y	
`INC/DEC (HL/IX+d/IY+d)`	Y	
`INC/DEC rr`	Y	
`INI, INIR, IND, INDR`	Y	
`JP nnnn`		Y
`JP cc,nnnn`		Y
`JP (HL/IX/IY)`	Y	
`LD (BC/DE),A`	Y	

TABLE 7. APPLICABILITY OF EXCEPTION SUFFIXES

Instruction (Class)	.S/.L	.IS/.IL
LD (HL/IX+d/IY+d),n	Y	
LD (HL/IX+d/IY+d),r	Y	
LD (HL/IX+d/IY+d),rr	Y	
LD (nnnn),r	Y	
LD (nnnn),rr [incl. IX, IY, SP]	Y	Y
LD A,(BC/DE)	Y	
LD r,(HL/IX+d/IY+d)	Y	
LD r,(nnnn)		Y
LD rr,(HL/IX+d/IY+d)	Y	
LD rr,(nnnn) [incl. IX, IY, SP]	Y	Y
LD rr,nnnn [rr other than SP]		Y
LD SP,HL/IX/IY	Y	
LD SP,nnnn	Y	Y
LDI, LDIR, LDD, LDDR	Y	
OUTI, OTIR, OUTD, OTDR	Y	
POP	Y	
PUSH	Y	
RET, RETI, RETN	Y	
RLC/RL/RRC/RR/SLA/SRA/SRL (HL/IX+d/IY+d)	Y	
RST	Y	
TST (HL)	Y	

A few instructions, that involve both a multi-byte register and a direct memory address or immediate datum, are affected by both exceptions. The proper suffix for exceptions on these instructions is .SIS, .SIL, .LIS, or .LIL.

For the sake of those instructions, the prefix bytes always express both kinds of exceptions. The prefix bytes replace several Z80 and Z80180 instructions that have no function. If an eZ80 assembler encounters one of these replaced instructions, it will issue a warning message and assemble it as a standard NOP (00H). Table 8 shows the eZ80 prefix bytes.

TABLE 8. NEW PREFIX BYTES ON THE EZ80

opcode	Z80 instruction	eZ80 prefix
40H	LD B,B	.SIS
49H	LD C,C	.LIS
52H	LD D,D	.SIL
5BH	LD E,E	.LIL

As for the traditional Z80 prefix bytes, the eZ80 does not allow an interrupt to occur between fetching one of these prefix bytes and fetching the following instruction. These prefix bytes must precede traditional Z80 prefix bytes.

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. Addresses in I/O space always have A23–16 all 0.

RESET CONDITIONS

The effects of Reset on 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.

S INSTRUCTION SET

Classes of Instructions

TABLE 9. 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

TABLE 10. 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 11. 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 12. 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 13. 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 14. BIT MANIPULATION INSTRUCTIONS

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

TABLE 15. BLOCK TRANSFER INSTRUCTIONS

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

TABLE 16. 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 17. 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

TABLE 18. 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