Application Note

Setting Interrupts with the eZ80® CPU

AN017001-0903

ZiLOG Worldwide Headquarters • 532 Race Street • San Jose, CA 95126
Telephone: 408.558.8500 • Fax: 408.558.8300 • www.ZiLOG.com

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters
532 Race Street
San Jose, CA 95126
Telephone: 408.558.8500
Fax: 408.558.8300
www.zilog.com

Information Integrity

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZiLOG Sales Office to obtain necessary license agreements.

Document Disclaimer

Table of Contents

List of Figures

List of Tables

Abstract

This Application Note describes how to set maskable interrupts for the each of the devices comprising ZiLOG's eZ80® and eZ80Acclaim!™ product lines in the ADL and Z80 memory modes. The document discusses how to relocate the interrupt vector table, map interrupt service routines in the interrupt vector table, and create interrupt service routines for processors operating in different memory modes.

The eZ80® product line includes the eZ80190 and eZ80L92 general-purpose microprocessors. The eZ80Acclaim!™ product line includes the eZ80F91, eZ80F92, and eZ80F93 Flash microcontrollers.

eZ80® CPU, Memory Modes, and Interrupts

This section presents a brief background on the eZ80® CPU, focusing on its memory modes and interrupts.

eZ80® CPU

The eZ80® CPU is ZiLOG's next-generation Z80 processor core. It is the basis of the new family of integrated microcontrollers, and includes the following features:

- Upward code-compatible from Z80 and Z180 products
- Several address-generation modes, including 24-bit linear addressing
- 24-bit registers and ALU
- 8-bit data path
- Single-cycle instruction fetch
- Pipelined fetch, decode, and execute

Memory Modes

The eZ80® CPU is capable of operating in two memory modes:

- Z80 Memory mode
- Address and Data Long mode

A description of each of these modes follows.

Z80 Memory mode. for backward compatibility with Z80 programs; the CPU operates in Z80 Memory mode with 16-bit addresses and 16-bit CPU registers. Z80 Memory mode is also called Z80 mode.

Address and Data Long mode. In ADL mode, the eZ80® CPU operates with 24-bit linear addressing and 24-bit CPU registers.

An ADL bit (0 for Z80 mode; 1 for ADL mode) controls memory mode selection.

eZ80® CPU Response to a Maskable Interrupt

The eZ80® CPU is capable of servicing a maskable interrupt using one of three interrupt modes: Interrupt Mode 0, Interrupt Mode 1, or Interrupt Mode 2. These modes are set by the IM0, IM1, or IM2 instructions, respectively. These interrupt modes provide backward compatibility with Z80 processors. However, not all products within the eZ80® family support all three interrupt modes.

The eZ80® line of microprocessors also supports vectored interrupts for on-chip peripherals. With vectored interrupts, the CPU fetches the low-order interrupt vector address from an internal vectored bus, IVECT [8:0]. The internal vectored bus is used exclusively for on-chip peripherals.

This Application Note focuses on setting maskable interrupts using Interrupt Mode 2. For detailed descriptions of interrupt modes and vectored interrupts, refer to the eZ80® CPU User Manual (UM0077).

The next section details the differences between the eZ80F91 device and the remainder of the devices in the eZ80® family, with respect to the I Registers and the IVECT Registers.

eZ80F91 vs. Other eZ80® Processors' Interrupt Registers

For all eZ80® processors, the interrupt controller routes interrupt request signals from the internal peripherals, the external devices (via the internal port I/O), and the nonmaskable interrupt (NMI) pin to the CPU.

On the eZ80F91 device, all maskable interrupts use the CPU's vectored interrupt function. The size of the Interrupt Page Address Register, or I Register, is 16 bits in the eZ80F91 device, differing from the other versions of the eZ80® CPU, to allow for a 16MB range of interrupt vector table placement. Additionally, the size of the IVECT Register in the eZ80F91 device is 9 bits (8 bits in the other eZ80® devices), to provide an interrupt vector table that can be expanded and is more easily integrated with other interrupts. Table 1 lists the interrupt vector addresses in ADL mode for each of the eZ80® devices.

Table 1. Interrupt Vector Address for eZ80® Devices in ADL Mode

eZ80® Device	Size of I Register	Size of IVECT Register	ISR Address (ADL Mode)
eZ80F91	16 bits	9 bits	{I[15:1], IVECT[8:0]}*
eZ80F92	8 bits	8 bits	{I[7:0], IVECT[7:0]}

Note: Only 15 bits of the I Register are used. The 16th bit is overwritten by the msb of the IVECT Register.

Table 1. Interrupt Vector Address for eZ80® Devices in ADL Mode (Continued)

eZ80® Device	Size of I Register	Size of IVECT Register	ISR Address (ADL Mode)
eZ80F93	8 bits	8 bits	{I[7:0], IVECT[7:0]}
eZ80L92	8 bits	8 bits	{I[7:0], IVECT[7:0]}
eZ80190	8 bits	8 bits	{I[7:0], IVECT[7:0]}

Note: Only 15 bits of the I Register are used. The 16th bit is overwritten by the msb of the IVECT Register.

Table 2 describes the interrupt vector address for the eZ80® devices in Z80 mode.

Table 2. Interrupt Vector Address for eZ80® Devices in Z80 Mode

eZ80® Device	Size of MBASE Register	Size of I Register	Size of IVECT Register	ISR Address (Z80 Mode)
eZ80F91	8 bits	8 bits	8 bits	{MBASE[7:0], I[7:0], IVECT[7:0]}
eZ80F92	8 bits	8 bits	8 bits	{MBASE[7:0], I[7:0], IVECT[7:0]}
eZ80F93	8 bits	8 bits	8 bits	{MBASE[7:0], I[7:0], IVECT[7:0]}
eZ80L92	8 bits	8 bits	8 bits	{MBASE[7:0], I[7:0], IVECT[7:0]}
eZ80190	8 bits	8 bits	8 bits	{MBASE[7:0], I[7:0], IVECT[7:0]}

The vectors are 4 bytes (32 bits) apart, even though only 3 bytes (24 bits) are required. The fourth byte is reserved for programmability and expansion purposes. Starting the interrupt vectors at address location 40h allows for easy implementation of the interrupt controller vectors with the Reset (RST) vectors.

In ADL mode, the full 24-bit interrupt vector is located at starting address {I[15:1], IVECT[8:0]}, where I[15:0] is the CPU's Interrupt Page Address Register (see Table 1).

In contrast to the eZ80F91 MCU, the other devices in the eZ80® family support an 8-bit I Register and an 8-bit IVECT Register, making the interrupt vectors 16 bits long (see Table 2).

Note: The ZDSII C Compiler supports only ADL mode; as a consequence, locating the interrupt vector table and creating the interrupt service routines can only be performed in the C coding language. In Z80 mode, these tasks are performed in the Assembly language.

To set interrupts in all eZ80® devices, the user must perform the following tasks:

- Locate the interrupt vector table

- Map the ISR location to the interrupt vector table
- Write an interrupt service routine or interrupt handler

The following sections describe how to perform these tasks in ADL and Z80 modes, including examples.

Setting Interrupts in ADL Mode

All maskable interrupts for the eZ80® family of devices use the eZ80® CPU's vectored interrupt function. The eZ80F91-based interrupt vector locations have a 24-bit address. The remainder of the eZ80® devices have a 16-bit address (see Table 1).

Relocating the Interrupt Vector Table

For the eZ80F91 Device. the interrupt vector table is constrained to start at the boundary of 512 bytes. For the eZ80F91 MCU, using the 15 bits of the I Register [15:1], the interrupt vector table can be mapped onto any of the 32,768 pages (16MB divided into pages of 512 bytes).

Figure 1 illustrates the boundaries where the interrupt vector table can be located. While relocating the interrupt vector table, the vector starting address must be placed at the beginning of the page address and not in the middle of the page. Examples of a starting address are `000000h`, `000200h`, and `000400h`.

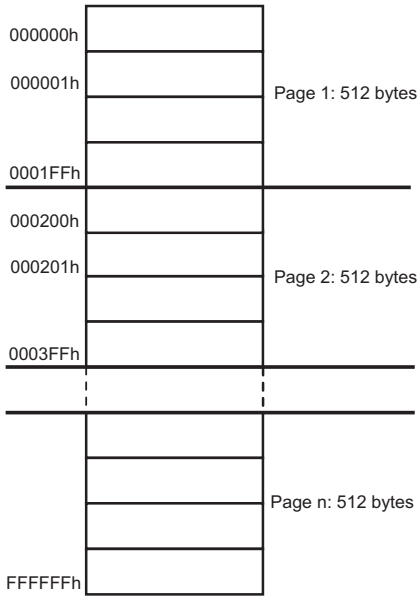

Figure 1. Interrupt Vector Table Boundaries for Relocation in eZ80F91 MCU

The following start-up code is an illustration of the kind of code that must be added to the user's existing `startup.asm` file when relocating the interrupt vector table for the eZ80F91 MCU.

In this example, the interrupt vector table is relocated to address location `FFE000h` within on-chip SRAM.

```
;*****************************************************************
The start-up code presented below defines an interrupt vector table for
the eZ80F91 MCU. It must be aligned to the 512-byte boundary of RAM.
;*****************************************************************
.assume ADL =1 ;This is an assembler directive
  NUM_VECTORS EQU 64          ;Initialize all of the interrupt vector
                              ;location
.def __vector table          ;
define __vectab, space=RAM, align= 512
.sect "__vectab"
ORG %FFE000                   ;Base address of the Interrupt vector.
                              ;The interrupt vector table is to be
                              ;relocated to ADDRESS 204800. This
                              ;address must be a multiple of 512.
                              ;The ORG directive is used to locate
                              ;the base address
__vector_table:
ds NUM_VECTORS*4              ;Each vector is a 4-byte address
                              ;pointing to the __vectptr segment.
                              ;The number of interrupts is 64 and
                              ;hence 256 bytes of memory is defined
                              ;Each vector takes up four bytes of
                              ;memory location.
;*****************************************************************
; The following start-up code sets the eZ80 CPU in interrupt mode 2 and
; loads the base address (FFE0 in this example) to the 16-bit I Register.
; Relocates the vector table.
;*****************************************************************
im 2 ; Interrupt mode 2
ld hl, __vector_table >> 8 & 0ffffh  ;
ld i,hl
;*****************************************************************
;*****************************************************************
```

In the following sample code, the interrupt vector table is relocated to address location `003E00h` on the on-chip Flash.

```
;*****************************************************************
The start-up code presented below defines an interrupt vector table for
the eZ80F91 MCU. It must be aligned to the 512-byte boundary of ROM.
```

```
;*******************************************************************
. assume ADL =1 ;This is an assembler directive
  NUM_VECTORS EQU 64          ;Initialize all of the interrupt vector
                              ;location
. def __vector table          ;
define __vectab, space=ROM, align= 512
.sect "__vectab"
ORG %003E00                   ;Base address of the Interrupt vector.
                              ;The interrupt vector table is to be
                              ;relocated to ADDRESS 003E00h This
                              ;address must be a multiple of 512.
                              ;The ORG directive is used to locate
                              ;the base address

__vector_table:               ;base address + the timer 0 interrupt
ORG %003E00+%54               ;vector address
dl_ISR_timer0                 ;The address of the timer0_isr is
XREF _ISR_timer0              ;mapped at the relocated vector table


;*******************************************************************
;The set_vector function cannot be used when mapping the interrupt
;service routines to the corresponding vector address.
;*******************************************************************
; The following start-up code sets the eZ80 CPU in interrupt mode 2 and
; loads the base address (003E in this example) to the 16-bit I Register.
; Relocates the vector table.
;*******************************************************************
im 2 ; Interrupt mode 2
ld hl, __vector_table >> 8 & 0ffffh  ;
ld i,hl
;*******************************************************************
*;*******************************************************************
**
```

For eZ80F92, eZ80F93, eZ80190, and eZ80L92 Devices. The relocation process noted above differs slightly for the remainder of the eZ80® family. For example, consider the TIMER0 (PRT 0) ISR, starting at the three-byte address location 204800h. The default TIMER0 interrupt vector location resides at 0Ah for the eZ80F92 MCU. Assuming that the interrupt vector table is relocated to start at address E000h, this start location points to location E114h, which contains a jump instruction to the TIMER0 ISR address. Figure 2 illustrates the default and relocated jump tables for the TIMER0 ISR location.

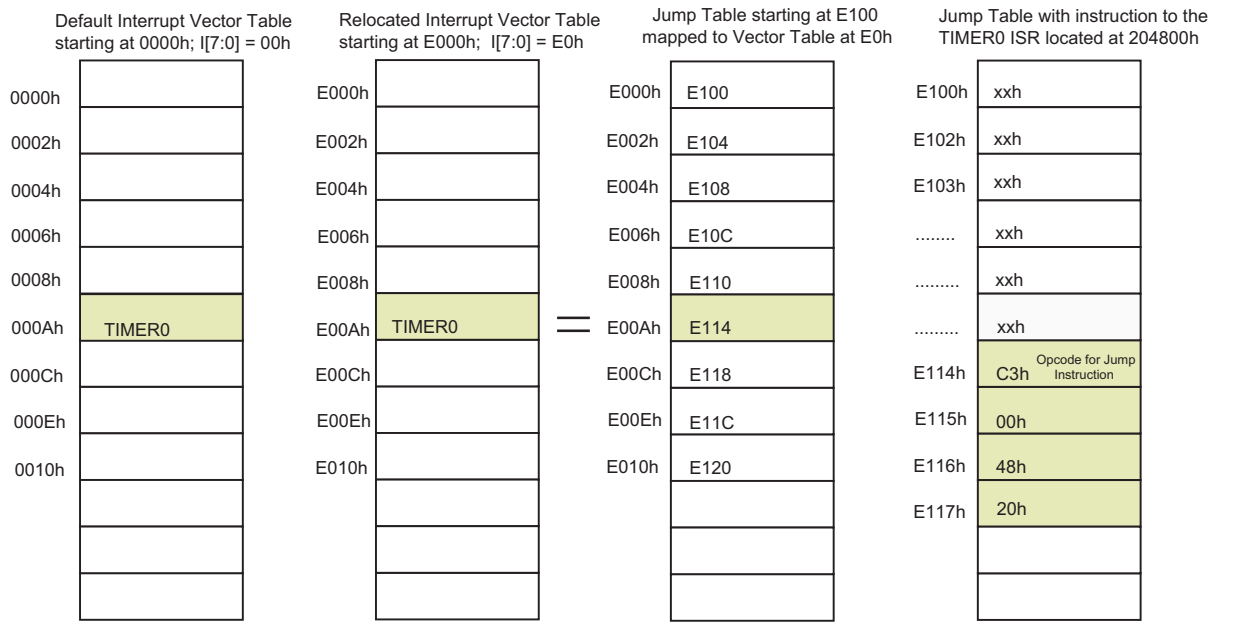

Figure 2. Memory Map Relocating Interrupt Vector Address with Jump Table

The following start-up code illustrates the type of code that must be added to the user's existing `startup.asm` file, when relocating the interrupt vector table using eZ80® devices other than the eZ80F91, with the 8-bit I Register and the 8-bit IVECT Register.

In this example, the interrupt vector table is relocated to address location `E000h` within on-chip SRAM.

```
;*****************************************************************
;Each interrupt vector is a 16-bit address pointing into the __vecptr
;segment. This segment must be aligned on a 256 byte boundary of RAM ;
;and must reside in the lower 64KB of memory.
;*****************************************************************
.assume ADL=1                          ;This is an assembler
;directive

RELOCATED_VECTOR_TABLE  EQU  E000h
NUM_VECTORS             EQU  128
. def __vector_table
define __vectab, space=RAM, align=256
. sect "__vectab"
ORG RELOCATED_VECTOR_TABLE
__vector_table:
```

```
ds NUM_VECTORS*2                        ;Each vector is a 2-byte
;address pointing into the
;__vectptr segment
;*****************************************************************
; This start-up code relocates the vector table from absolute 0000h
; location to E000h location. It loads I Registers with the value E0h.
;*****************************************************************
im 2 ; Interrupt mode 2
ld, __vector_table >> 8 & 0ffh
ld i, a ; Load interrupt vector base
;*****************************************************************
*;*****************************************************************
```

The following code is an illustration of the kind of code that can be added to the user's existing `startup.asm` file to locate the jump table within the 64KB memory space, when using eZ80® devices other than the eZ80F91 MCU.

In the following sample code, the jump table is relocated to address location `E100h` within on-chip SRAM.

```
;*****************************************************************
;Define jump table. Each entry is a JP.LIL to an interrupt handler.
;This segment must reside in the lower 64KB of RAM.
;*****************************************************************
RELOCATED_JUMP_TABLE EQU E100h
define __jumptab,space=RAM
.sect "__jumptab"             ; __vectors is predefined
ORG RELOCATED_JUMP_TABLE
__jump_table:
ds NUM_VECTORS*4              ; Each entry is a JP.LIL to a handler
;*****************************************************************
*;*****************************************************************
```

The following lines of code illustrate how to map the jump table to the relocated vector table.This start-up code must be added to the user's existing `startup.asm` file.

```
;*****************************************************************
ld hl,__vector_table
ld b,NUM_VECTORS
ld iy,__jump_table
$1:
ld.sis (hl),iy                ; store vector
inc hl
inc hl                        ; next vector address
lea iy,iy+4                   ; next jp.lil address
```

```
dec b
jr nz,$1
;*****************************************************************
*;*****************************************************************
```

The following start-up code illustrates the type of code that must be added to the user's existing `startup.asm` file when relocating the interrupt vector table and using the 8-bit I Register and the 8-bit IVECT Register on eZ80® devices other than eZ80F91.

In the following sample code, the interrupt vector table is relocated to address location `3E00h` in on-chip Flash memory.

```
;*****************************************************************
; Each interrupt vector is a 16-bit address pointing into the __vecptr
; segment. This segment must be aligned on a 256 byte boundary of ROM ;
;and must reside in the lower 64KB of memory.
;*****************************************************************
.assume ADL=1                          ; This is an assembler directive

RELOCATED_VECTOR_TABLE  EQU  3E00h
NUM_VECTORS             EQU  128
. def __vector_table
define __vectab, space=ROM, align=256
. sect "__vectab"
ORG RELOCATED_VECTOR_TABLE
__vector_table:

dw __jump_table, __jump_table+2,------------ __jump_table+254

;*****************************************************************
; This start-up code relocates the vector table from absolute 0000h
; location to E000h location. It loads I Registers with the value E0h.
;*****************************************************************
im 2 ; Interrupt mode 2
ld, __vector_table >> 8 & 0ffh
ld i, a ; Load interrupt vector base
;*****************************************************************
*;*****************************************************************
```

The following code is an illustration of the kind of code that can be added to the user's existing `startup.asm` file to locate the jump table within the 64KB memory space, when using the eZ80® devices other than the eZ80F91 MCU.

In the following sample code, the jump table is relocated to address location `3F00h` in on-chip Flash memory.

```
;*********************************************************************
;Define jump table. Each entry is a JP.LIL to an interrupt handler.
;This segment must reside in the lower 64KB of ROM.
;*********************************************************************
RELOCATED_JUMP_TABLE EQU E100h
define __jumptab,space=ROM
.sect "__jumptab"                          ; __vectors is predefined
ORG RELOCATED_JUMP_TABLE
__jump_table:

ORG RELOACATED_JUMP_TABLE+%0A              ;%0A is the address of the
                                           ;timer0 interrupt vector
Db      %C3;Op Code for jump
.trio _ISR_timer0                          ;maps the address of the
                                           ;timer0_isr
XREF _ISR_timer0                           ;to relocated jump table
;*********************************************************************
*;*********************************************************************
```

Mapping the ISR Location in the Interrupt Vector Table

The maximum addressable capacity of eZ80® devices is 16MB.

Note: ZiLOG recommends that the I Register value for eZ80® devices be changed from its default value of 00h to avoid conflict between the NMI, RST instruction addresses, and the maskable interrupt vectors.

For the eZ80F91 Device. All of the interrupt vector addresses can be located anywhere in the 16MB address space, barring the locations where the non-maskable interrupt (NMI) is located (066h) and the locations between RST0 to RST8 and RST38, which are all absolute addresses. The user's program must store the ISR's starting address in a four-byte interrupt vector location.

For example, consider a TIMER0 interrupt service routine that starts at three-byte address location 123456h. The default TIMER0 interrupt vector location for the eZ80F91 MCU is at 054h and the TIMER0 interrupt service routine's address is therefore mapped as follows:

```
{I Register [15:1], 054h}   ------------>   56h
{I Register [15:1], 055h}   ------------>   34h
{I Register [15:1], 056h}   ------------>   12h
{I Register [15:1], 057h}   ------------>   NOT USED
```

Note: The address location {I Register [15:1], 057h} is not used. The least significant byte is stored at the lower address per the Little Endian format.

For eZ80F91 interrupts, the full 24-bit interrupt vector is located at starting address {I [15:1], IVECT [8:0]; the lower bit of the I Register (bit 0) is overwritten with the most significant bit (msb) of the IVECT Register. Setting bit 0 of the I Register has no effect on the interrupt vector locations.

For eZ80F92/F93/190/L92 Devices. The interrupt vectors must be located within the 64KB address space for the remainder of the eZ80® family. Because the interrupt vector location can take only two-byte addresses (see Table 1), the ISR must also be located within the same 64KB address space. However, by using a jump table, the ISR can be located anywhere in the 16MB address space. The jump table, however, must be located within the 64KB memory space where the interrupt vectors are also located.

As an example, the default TIMER0 (PRT 0) interrupt vector for the eZ80190 MPU is located at `06h`. Therefore, the TIMER0 interrupt service routine's address—`123456h`, is mapped as follows:

```
{I Register [7:0], 06h}  ------------>   56h
{I Register [7:0], 07h}  ------------>   34h
{I Register [7:0], 08h}  ------------>   12h
{I Register [7:0], 09h}  ------------>   NOT USED
```

Writing the Interrupt Service Routine

To create an interrupt service routine in ADL mode, the `interrupt` keyword is used. This keyword is a storage class that is applicable only to functions. Alternatively, a keyword combination of `#pragma interrupt` can be used.

For example, to write an interrupt service routine for TIMER0, use either of the two code segments presented below.

```
interrupt void ISR_Timer0 (void)
{
  unsigned char temp= 10;
  ....;
  ....;
}
```

or

```
#pragma interrupt
void ISR_Timer0 (void)
{
 unsigned char temp = 10;;
 ....;
 ....;
```

```
}
```

The _set_vector function is used to attach an interrupt service routine (which is a C function) to an interrupt vector. The _set_vector routine takes two arguments—the first is an integer defining the interrupt number, and the second is the name of the associated interrupt service routine.

For the eZ80F91 Device. The following code illustrates how the _set_vector routine is called.

```
# define VECTOR_TIMER0 0x54   // Vector offset value as mentioned in
                              // interrupt vector table for F91
# include <ez80.h>

void ISR_TIMER0();            // Function prototype declaration
void  Init_TIMER0 (void);     // Function prototype declaration

void set_vector(unsigned short int, void(*handlr)(void);

main( )
{
  _di(  );                    // disable interrupts
   Init_TIMER0;               // Initialize timer0
  _ei( );1                    // enable interrupts

}

Init_TIMER0 ( )
{
  _set_vector(VECTOR_TIMER0, ISR_Timer0);
  Initialize timer0
  ...;
  ...;
}
```

The Init_TIMER0() function calls the _set_vector function. The TIMER0 interrupt vector address is located at 054h. This address is passed to the _set_vector function along with the address of the ISR, ISR_TIMER0. The _set_vector function is written in assembly and is defined in the startup.asm file. The code snippet provided below is for the _set_vector function.

The following start up code must be added in the existing startup.asm file.

```
;*****************************************************************
; The address of the interrupt handler (interrupt service routine) is
; copied into corresponding 3-byte vector address location.
```

```
;
; void _set_vector (unsigned short vector, void (*hndlr)(void));
;*********************************************************************
*  . def    __set_vector
__set_vector:
  ld      iy,0
  add     iy, sp
  ld  hl,(iy+3)
  ld bc,__vector_table
  add     hl,bc
  ld      bc, (iy+6)
  ld      (hl), bc
  ret
;*********************************************************************
*;*********************************************************************
```

For the eZ80F92, eZ80F93, eZ80190, and eZ80L92 Devices. The following sample code illustrates how the `_set_vector` function is called for the remainder of the eZ80® devices.

```
# define TIMER0 0x14          // Vector offset value for TIMER0
      //(PRT 0)as mentioned in vector table
      // for eZ80F92 * 2 (0Ah * 2)
      // Vector offset for TIMER0 (PRT 0)
      // for eZ80190 is 06h (06h * 2)

# include <ez80.h>

void ISR_TIMER0();            // Function prototype declaration
void  Init_TIMER0 (void);     // Function prototype declaration

void set_vector(unsigned short int, void(*handlr)(void);

Init_TIMER0 ( )
{
  _set_vector(TIMER0, ISR_Timer0);
  Initialize TIMER0;

  ....;
  ....;
}
```

The following start-up code must be added to the user's existing `startup.asm` file.

```
;*********************************************************************
```

```
;Define __set_vector to install a user interrupt handler
;
;void _set_vector(unsigned short vector, void (*hndlr)(void));
;
;
;Argument1 - address of user interrupt handler
;Argument2 - define  TIMER0  0x14 (for eZ80F92)
;
;*****************************************************************
  .def __set_vector
  __set_vector:
  ld ix, 0
   add     ix, sp
   ld      hl,0; Clear UHL
   ld.sis  hl, (ix+3); Vector offset
   ld bc, RELOCATED_JUMP_TABLE

   add      hl, bc; hl is address of jp
   ld       (hl), %C3; Op Code for jump
   inc      hl; hl is address of handler
   ld       bc, (ix+6) handler
   ld       (hl), bc; store new vector address
   ret

;*****************************************************************
*;*****************************************************************
```

Setting Interrupts in Z80 Mode

In eZ80F91-based applications that run exclusively in Z80 mode, the interrupt vector address is {MBASE, I[7:1], IVECT[8:0]}. For other eZ80® CPU–based applications, the interrupt vector address is {MBASE, I[7:0], IVECT[7:0]}. A 16-bit word is fetched from the interrupt vector address and loaded into the lower two bytes of the Program Counter, PC [15:0].

In Z80 mode, the upper byte of the I Register bits, [15:8], is not used.

Because the ZDSII C compiler does not support Z80 mode, locating the interrupt vector table and writing the interrupt service routine must be performed in Assembly language.

Relocating the Interrupt Vector Table

For the eZ80F91 Device. The start-up code presented below locates the interrupt vector table for the eZ80F91 device, in Z80 mode. Each entry is a 16-bit address pointing into the `__vecptr` segment. This segment must be aligned on

the 512-byte page boundary of RAM and must reside in the lower 64KB of memory.

```
;*****************************************************************
;*****************************************************************
. assume ADL = 0
NUM_VECTORS     EQU     64            ; Initialize all of the interrupt
                                      ; vector location
. def  __vector table                 ;
define __vectab, space=RAM, align= 512
.sect  "__vectab"
ORG %0400                             ;Base address of the
                                      ;Interrupt vectors
;
;*****************************************************************
; If the interrupt vector table is to be mapped at ADDRESS 2048, use the
; ORG directive as mentioned. However, this address should be a multiple
; of 512.
;*****************************************************************
; Set the interrupt to mode 2; load the interrupt base address to 8 bit I
; register. In this example, 04h is moved to I Register. At the beginning
; of the program, the user should take care of initializing MBASE
; register with an appropriate value.
;*****************************************************************
im 2                                  ;Interrupt mode 2
ld a, __vector_table >> 8 & 0ffh      ;Difference to ADL mode is
                                      ;highlighted
ld i,a                                ;Load interrupt vector base
;*****************************************************************
```

For the eZ80F92, eZ80F93, eZ80190, and eZ80L92 Devices. The start-up code presented below defines the interrupt vector table for the remainder of the eZ80® devices, in Z80 mode. Each entry is a 16-bit address pointing into the `__vector` segment. This segment must be aligned to the 256 byte boundary of RAM and must reside in the lower 64KB of memory.

```
;*****************************************************************
;*****************************************************************
. assume ADL = 0;
NUM_VECTORS     EQU     64    ; Initialize all of the interrupt vector
                              ; location
. def  __vector table ;
define __vectab, space=RAM, align= 256
.sect  "__vectab"
ORG    %0300                  ; Base address of the Interrupt
```

```
                                ; vectors.
;*********************************************************************
;The interrupt vector table is mapped to ADDRESS  0300h; the ORG
;directive is used. However, this address should be the multiple of 256.
;*********************************************************************
;Set the interrupt to mode 2; load the interrupt base address to 8 bit I
;register. In this example, 03h is moved to I register. At the beginning
;of the program, the user should take care of initializing  MBASE
;register with an appropriate value.
;*********************************************************************

im 2                                    ; Interrupt mode 2
ld a, __vector_table >> 8 & 0ffh        ;
ld i,a                                  ; Load interrtup vector base
;*********************************************************************
*;*********************************************************************
```

Note: The jump table concept for relocating the interrupt vector table cannot be applied in Z80 mode.

Mapping the ISR location in the Interrupt Vector Table

For the eZ80F91 Device. Using TIMER0 as an example, let us assume that its interrupt service routine resides at the two-byte address location 1234h. The TIMER0 interrupt vector location for the eZ80F91 device is at 054h. The TIMER0 interrupt service routine's address is stored as follows:

```
{MBASE, I Register [7:1], 054h}   ------------>   34h
{MBASE, I Register [7:1], 055h}   ------------>   12h
{MBASE, I Register [7:1], 056h}   ------------>   Not used
{MBASE, I Register [7:1], 057h}   ------------>   Not used
```

The address locations {MBASE, I Register [7:1], 056h} and {MBASE, I Register [7:1], 057h} are not used.

For the eZ80F92, eZ80F93, eZ80190, and eZ80L92 Devices. The TIMER0 (PRT 0) interrupt vector location for the eZ80F92 device is at 0Ah. Assuming that its interrupt service routine resides at the two-byte address location 1234h, the TIMER0 interrupt service routine's address for the eZ80F92 device is stored as follows:

```
{MBASE, I Register [7:1], 0XXh}   ------------>   34h
{MBASE, I Register [7:1], 0XXh}   ------------>   12h
```

Writing the Interrupt Service Routine

The example code below illustrates how to write an interrupt service routine in Z80 mode for all of the eZ80® devices.

```
_ISR_Timer0:
  DI;
  EXX
  EX AF, AF’
------
------
------
------
EXX
EX AF, AF’
EI;
RETI
```

For the eZ80F91 Device. The following assembly code illustrates how to load the `ISR_Timer0` address at the TIMER0 interrupt vector location for the eZ80F91 device in Z80 mode. This code can be added to the user generated assembly program.

```
VECTOR_TIMER0 EQU %054  // Vector offset for the
                                        // TIMER0 ISR for eZ80F91 is
                                        // 054h.

ld hl, VECTOR_TIMER0; Timer0
ld bc,__vector_table
add     hl,bc
ld      bc, _ISR_Timer0
ld      (hl), bc
```

For the eZ80F92, eZ80F93, eZ80190, and eZ80L92 Devices. The following code illustrates how to load the `ISR_TIMER0` address at the eZ80F92 MCU’s TIMER0 (PRT 0) interrupt vector location at `0Ah`. This code can be added to the user generated assembly program.

```
TIMER0 EQU %0A    // Vector offset for TIMER0(PRT 0)
                                        // for eZ80F92 is 0Ah
ld hl, VECTOR_TIMER0; Timer0
ld bc,__vector_table
add     hl,bc
ld      bc, _ISR_Timer0
ld      (hl), bc
```

Appendix A—References

Further details about the eZ80® family of products can be found in the references listed in Table 1.

Table 1. List of References

Topic	Document Name
eZ80® CPU	eZ80® CPU User Manual (UM0077)
eZ80F91 MCU	eZ80F91 Product Specification (PS0192)
eZ80F92 and eZ80F93 MCUs	eZ80F92/eZ80F93 Product Specification (PS0153)
eZ80190 MPU	eZ80190 Product Specification (PS0066)
eZ80L92 MPU	eZ80L92 MPU Product Specification (PS0130)