Z80 Journal

Z80 Interrupts

Posted on April 15, 2015

The Z80 microprocessor supports three interrupts modes; 0, 1, and 2. In interrupt mode 0 the Z80 gets an instruction from the data bus given by the peripheral and executes it. This instruction is normally RST0 -> RST7 which resets the Z80 to a specific location given by that instruction. In interrupt mode 1 the Z80 jumps to address 0038h where it runs a routine that the programmer implements. In interrupt mode 2 the Z80 gets a byte from the data bus and jumps to the 16-bit address formed by combining the 'I' (interrupt vector) register and the supplied byte from the peripheral.

Since most peripherals don't include the functionality to automatically respond to the interrupt conditions that the Z80 requires I would not recommend implementing interrupt modes 0 and 2 in a simple system. If you are only using the Z80 family peripherals then interrupt mode 2 is specifically tailored to you, and it may be worth looking into. If you are not using these peripherals, and you want a simpler interrupt structure then choose interrupt mode 1.

As stated above in interrupt mode 1 the Z80 jumps to address 0038h where is runs a routine that the programmer implements. There are a few conditions that need to be met before this happens however.

1. Interrupts must be enabled. (EI instruction)
2. Your peripheral must be connected to the /INT input on the Z80.
3. An interrupt routine must be programmed at the correct place in memory.

The interrupt routine must also follow a few guidelines for correct function. The routine should save all registers either by exchanging them with shadow registers using EX and EXX, or by pushing them onto the stack. Before the routine finishes all of the working registers should be restored by again, either exchanging them back with the shadow registers using EX and EXX, or by popping them off the stack if they were pushed on before. Interrupts should be enabled with the EI instruction, and RET(I) should be used to exit the interrupt routine. RETI (return from interrupt) has exactly the same functionality as RET to return by popping the return address off the stack. The difference between the two is that RETI affects internal flags in Z80 peripherals that are specifically for the daisy-chained peripheral interrupt scheme. This interrupt scheme is specific to only Z80 peripherals which I am not using, so I stick with using RET instead.

An example of a mode 1 routine can be seen below,

```
;Interrupt mode 1
.org $0038

;Disable interrupts
DI
```

```
EXX

;Start interrupt routine here
LD A, (interrupt_counter)
INC A
LD (interrupt_counter), A
CALL interrupt_service_routine
;End interrupt routine here

;Restore register states
EXX
EX AF,AF'

;Enable interrupts
EI

RET
```

The above example will be triggered when our peripheral activates (pulls logic-low) the Z80 /INT pin. Pulling this pin low activates the mask-able interrupt that was selected. By default interrupt mode 1 is implemented, but for good practice you should manually enable interrupt mode 1 by using the instruction IM1 very early in your code.

Since the interrupt address location occurs at a very early address 0038H the programmer should avoid putting code around this location. To alleviate this problem my first instruction jumps over this location to the MAIN routine.

You can see this in the following code snippet,

```
;*****************************************************************************
;  PROGRAM:         Z80 mode 1 interrupts
;  PURPOSE:         Tests that mode 1 interrupts function properly
;  ASSEMBLER:       TASM 3.2
;  LICENCE:         The MIT Licence
;  AUTHOR :         MCook
;  CREATE DATE :    14 Apr 15
;*****************************************************************************

;ROM : 0000H - > 07FFH
;RAM : 8000H - > 87FFH

.ORG 00H

START:
            JP MAIN  ;Jump to the MAIN routine

.ORG 100H

RAMTOP:                 .EQU    87FFH   ;Top address of RAM
INTERRUPT_COUNTER:      .BYTE   0       ;Interrupt counter default value 0

MAIN:
            LD      SP,RAMTOP
            EI      ;Enable interrupts
            IM 1    ;Use interrupt mode 1
            HALT

;Interrupt mode 1
.ORG 38H

MODE1_INTERRUPT:
            ;Disable interrupts
            DI

            ;Save register states
            EX      AF,AF'
            EXX

            ;Start interrupt routine here
            LD      A,(INTERRUPT_COUNTER)
            INC     A
            LD      (INTERRUPT_COUNTER),A
            ;End interrupt routine here

            ;Restore register states
            EXX
            EX      AF,AF'

            ;Enable interrupts
            EI

            RET


.END
```

This section of code was assembled using the TASM assembler v3.2. It illustrates how to place the interrupt routine at 0038H while skipping over it to initially start the program at our main routine. Since the Z80 always starts at address 0000H for the first instruction we immediately place a jump instruction at this location that skips over 0038H so that we can still define the interrupt routine at the 0038H location.

The only problem with mode 1 interrupts is that the device cannot dictate the interrupt routine to use. In mode 1 the same interrupt routine will be run regardless of the peripheral that issues the interrupt by pulling the /INT pin logic-low. To deal with the problem the programmer needs to properly check each devices' status and branch within the interrupt routine to service the interrupting peripheral in the correct way. For instance, the 16550 UART contains an interrupt identification register that can be checked during the interrupt routine to provide the correct service for the issued interrupt.

Even though they are a bit convoluted to implement, interrupts provide a great platform to respond to peripheral events in your system. Without interrupts you would need to rely on polling to interact with peripherals which

Share this:

Twitter Facebook

Like

Be the first to like this.

About Matt Cook
Computer Engineer
View all posts by Matt Cook →

This entry was posted in Uncategorized and tagged Assembly, Code, Interrupts, Z80, Z80 Microprocessor. Bookmark the permalink.

Z80 Journal

Blog at WordPress.com.