1. Here's an example of assembly language programming for handling hardware interrupts (external interrupts) in the 8051 microcontroller:

```
ORG 0x0000; Reset vector
                ; Interrupt vector table
ORG 0x0003;  Address for external interrupt 0 (INT0)
LJMP INT0_ISR;  Jump to INT0 ISR
ORG 0x0013;  Address for external interrupt 1 (INT1)
LJMP INT1_ISR;  Jump to INT1 ISR
; Interrupt service routines
INT0_ISR:
    ; Your code here for INT0 ISR
    ; This code will be executed when an interrupt is triggered on INT0 pin
    ; You can perform any required tasks or operations
    RET0; Return from interrupt
INT1_ISR:
    ; Your code here for INT1 ISR
    ; This code will be executed when an interrupt is triggered on INT1 pin
    ; You can perform any required tasks or operations
    RETI; Return from interrupt
; Main program
ORG 0x0100; Start address of the main program
MAIN:
    ; Initialize interrupt-related registers
    MOV IE, #0x85; Enable INT0 and INT1 interrupt
    ; Your main program code here
```

```
    SJMP MAIN;  Infinite loop

END
```

In this example, we have used the same interrupt vector table as in the previous example, which is located at addresses 0x0003 and 0x0013 for INT0 and INT1, respectively.

1. To handle the hardware interrupts, we need to initialize the interrupt-related registers. In this example, we use the MOV instruction to set the IE (Interrupt Enable) register to enable INT0 and INT1 interrupts. The value 0x85 is used to enable INT0 (bit 0) and INT1 (bit 1) while keeping other interrupts disabled.

2. The INT0_ISR and INT1_ISR routines are the interrupt service routines for INT0 and INT1 interrupts, respectively. These routines are executed when an interrupt is triggered on the corresponding pin. You can write your code within these routines to perform the necessary tasks or operations. After executing the ISR code, we use the RETI instruction to return from the interrupt and resume the main program.

3. The main program, represented by the MAIN label, is where you can write your application code. Similar to the previous example, we have an infinite loop using the SJMP instruction to keep the program running indefinitely.

Again, please note that this is a basic example, and you should consult the datasheet and reference manual of your specific microcontroller to understand the interrupt handling process and configure the interrupt-related registers properly.

2.Here's an example of assembly language programming for handling Timer 0 (TF0) and Timer 1 (TF1) interrupts in the 8051 microcontroller:

```
ORG 0x0000; Reset vector

; Interrupt vector table

ORG 0x000B; Address for Timer 0 interrupt (TF0)

    LJMP TF0_ ISR; Jump to TF0 ISR

ORG 0x001B; Address for Timer 1 interrupt (TF1)

    LJMP TF1 _ISR;   Jump to TF1 ISR

; Interrupt service routines

TF0_ISR:

    ; Your code here for TF0 ISR

    ; This code will be executed when Timer 0 interrupt occurs

    ; You can perform any required tasks or operations
```

```
    RETI; Return from interrupt
TF1_ISR:
    ; Your code here for TF1 ISR
    ; This code will be executed when Timer 1 interrupt occurs
    ; You can perform any required tasks or operations
    RETI; Return from interrupt
; Main program
```

3. Here's an example of assembly language programming for handling software interrupts inthe 8051 microcontroller:

```
ORG 0x0000; Reset vector
; Interrupt vector table
ORG 0x0023; Address for software interrupt (SWI)
    LJMP SWI_ISR;  Jump to SWI ISR
; Interrupt service routine
SWI_ISR:
    ; Your code here for SWI ISR
    ; This code will be executed when a software interrupt is triggered
    ; You can perform any required tasks or operations
    RETI;  Return from interrupt
; Main program
ORG 0x0100; Start address of the main program
MAIN:
    ; Your main program code here
    ; Trigger software interrupt
    LCALL TRIGGER_SWI
    SJMP MAIN;  Infinite loop
; Subroutine to trigger software interrupt
```

```
TRIGGER_SWI:
    ; Set the software interrupt flag
    SETB SWI_FLAG
    ; Trigger the interrupt by setting the appropriate interrupt bit
   SETB IE.4
    RET
; Variables
SWI_FLAG:    DS 1;  Software interrupt flag
END
```

In this example, we define the software interrupt vector in the interrupt vector table at address 0x0023. This is the location where the microcontroller will jump when a software interrupt is triggered.

1. We then define the interrupt service routine SWI_ISR, where you can write the code that needs to be executed when the software interrupt occurs. After executing the ISR code, we use the RETI instruction to return from the interrupt and resume the main program.
2. The main program, represented by the MAIN label, is where you can write your application code. In this example, we have an infinite loop using the SJMP instruction to keep the program running indefinitely.
3. To trigger a software interrupt, we have a subroutine TRIGGER_SWI that sets the software interrupt flag (SWI_FLAG) and sets the corresponding interrupt bit (IE.4). You can call this subroutine whenever you want to trigger the software interrupt in your program.

Please note that the software interrupt mechanism in the 8051 microcontroller may vary depending on the specific variant or manufacturer. The above example assumes a generic implementation of the software interrupt mechanism. Make sure to consult the datasheet and reference manual of your specific microcontroller to understand the software interrupt handling process and register configurations properly.

```
ORG 0x0100; Start address of the main program

MAIN:

    ; Initialize Timer 0 and Timer 1

    MOV TMOD, #0x11; Set Timer 0 and Timer 1 in mode 1 (16-bit mode)

    MOV TH0, #0x00;  Initialize Timer 0 high byte

    MOV TL0, #0x00 ; Initialize Timer 0 low byte

    MOV TH1, #0x00  ; Initialize Timer 1 high byte

    MOV TL1, #0x00  ; Initialize Timer 1 low byte

    ; Enable Timer 0 and Timer 1 interrupt

    SETB ET0; Enable Timer 0 interrupts

    SETB ET1; Enable Timer 1 interrupts

    SETB EA;  Enable global interrupts

    ; Start Timer 0 and Timer 1

    SETB  TR0;  Start  Timer  0

    SETB TR1;  Start Timer 1

    ; Your main program code here

    SJMP MAIN;  Infinite loop

END
```

In this example, we have used the Timer 0 and Timer 1 interrupts, which are TF0 and TF1 respectively, to generate periodic interrupts.

4. We define the Timer 0 interrupt vector at address 0x000B and the Timer 1 interrupt vector at address 0x001B in the interrupt vector table. These are the locations where the microcontroller will jump when the corresponding timer interrupts occur.

5. The interrupt service routines TF0_ISR and TF1_ISR are executed when Timer 0 and Timer 1 interrupts occur, respectively. You can write your code within these routines to perform the necessary tasks or operations. After executing the ISR code, we use the RETI instruction to return from the interrupt and resume the main program.

6. The main program, represented by the MAIN label, is where you can write your application code. In this example, we initialize Timer 0 and Timer 1 using the TMOD register, set the initial values for the timers using TH0, TL0, TH1, and TL1, and enable the Timer 0 and Timer 1 interrupts using ET0 and ET1 respectively. We also enable global interrupts using EA.

7. Finally, we start Timer 0 and Timer 1 by setting the TR0 and TR1 bits respectively.

Please note that the exact register names and values may vary depending on the specific variant or manufacturer of the 8051 microcontroller you are using. Make sure to consult the datasheet and reference manual of your specific microcontroller to understand the timer configuration and interrupt handling process properly.