Counter Programming in 8051 micro controller

In the context of 8051 microcontrollers, "counter programming" refers to the manipulation and control of the built-in hardware counters or timers available in the 8051 architecture. The 8051 microcontroller family typically includes one or more timer/counter modules, which can be utilized for various timing, counting, or event-driven operations.

The 8051 microcontroller has two standard counter/timer modules: Timer 0 (T0) and Timer 1 (T1). These modules can be configured and programmed to perform a range of tasks, such as generating precise time delays, measuring external events or pulses, generating PWM signals, or generating interrupts at specific intervals.

To perform counter programming in an 8051 microcontroller, you typically follow these steps:

1. **Configure the timer/counter module:** Set the control registers of the desired timer/counter module (T0 or T1) to configure its mode of operation, such as 16-bit or 8-bit mode, operating frequency, and other relevant settings.

2. **Set the initial value:** Load the initial value into the timer/counter register to start counting from a specific value. This value depends on the desired timing or counting requirements.

3. **Enable interrupts (if needed):** If you want to generate interrupts at specific intervals, enable the interrupt flag and corresponding interrupt enable bit for the timer/counter module.

4. **Start the timer/counter:** Enable the timer/counter module to start counting by setting the appropriate control bits.

5. **Handle interrupts (if used):** If interrupts are enabled, write the interrupt service routine (ISR) to handle the interrupt requests generated by the timer/counter module. The ISR can perform specific actions based on the interrupt source, such as updating variables, generating output signals, or executing other program logic.

6. **Perform required operations:** Depending on the application, you may need to monitor the counter value, check for specific conditions, or perform other actions based on the timer/counter status.

7. **Stop or reset the counter (if needed):** When the desired timing or counting operation is complete, you can stop the timer/counter or reset it to its initial value.

By effectively programming and utilizing the counter/timer modules in an 8051 microcontroller, you can perform precise timing operations, control external events, generate PWM signals, measure pulse durations, and much more. The specific details and capabilities of counter programming in the 8051 microcontroller may vary depending on the specific variant or model of the microcontroller you are working with and the associated development tools or programming language used.

Here are a few examples of counter programming in the 8051 microcontroller using Timer 0 (T0) and Timer 1 (T1):

Example 1: Generating a Delay

MOV TMOD, #01	; Set Timer 0 in 16-bit mode
MOV TH0, #0xF8	; Load initial value for Timer 0 high byte
MOV TL0, #0xCC	; Load initial value for Timer 0 low byte
SETB TR0	; Start Timer 0
DELAY	: Delay loop
JNB TF0, DELAY	; Wait until Timer 0 overflows (TF0 flag is set)
CLR TR0	; Stop Timer 0
CLR TF0	; Clear Timer 0 overflow flag

This program sets up Timer 0 in 16-bit mode and initializes its high and low bytes to create a delay of approximately 100 ms. The program enters a loop and waits until Timer 0 overflows (TF0 flag is set) before proceeding.

Example 2: Generating a PWM Signal

MOV TMOD, #10	; Set Timer 1 in 8-bit mode (for PWM)
MOV TH1, #0x80	; Load initial value for Timer 1
SETB TR1	; Start Timer 1
PWM:	; PWM loop
MOV P1, #0x00	; Output logic low on P1
ACALL DELAY	; Delay subroutine
MOV P1, #0xFF	; Output logic high on P1

```
ACALL DELAY              ; Delay subroutine
SJMP PWM                  ; Repeat PWM loop
DELAY:                    ; Delay subroutine
MOV R1, #0xFF            ; Initialize delay counter
DELAY_LOOP:
DJNZ R1, DELAY_LOOP      ; Decrement counter and repeat until zero
RET
```

This program sets up Timer 1 in 8-bit mode to generate a simple PWM signal on Port 1 of the microcontroller. The program alternates between logic low and logic high levels on Port 1 with a specific delay in between, achieved by a delay subroutine.

Programming Timer 0 and Timer 1 in 8051 examples

Example 1: Timer 0 Interrupt-based Timer

```
ORG 0x0000   ; Start of program memory
MOV TMOD, #01             ; Set Timer 0 in 16-bit mode
MOV TH0, #0x00            ; Load initial value for Timer 0 high byte
MOV TL0, #0x00           ; Load initial value for Timer 0 low byte
SETB ET0                 ; Enable Timer 0 interrupt
SETB EA                  ; Enable global interrupts
MOV TR0, #1              ; Start Timer 0
MAIN_LOOP:                ; Main program loop
    ; Your main program logic here
    SJMP MAIN_LOOP
TIMER0_ISR:               ; Timer 0 interrupt service routine
    ; Your Timer 0 interrupt code here
    RETI                        ; Return from interrupt
```

In this example, Timer 0 is set in 16-bit mode. The ET0 bit is set to enable Timer 0 interrupt, and the EA bit is set to enable global interrupts. The program then starts Timer 0, and the main program logic executes in a loop. Whenever Timer 0 overflows, it triggers an interrupt, and the microcontroller jumps to the TIMER0_ISR label to execute the Timer 0 interrupt service routine. After the ISR is complete, the program returns from the interrupt using the RETI instruction.

Example 2: Timer 1 with Capture and Compare Mode

```
ORG 0x0000              ; Start of program memory
MOV TMOD, #20           ; Set Timer 1 in capture and compare mode
SETB TR1                ; Start Timer 1
MAIN_LOOP:              ; Main program loop
    ; Your main program logic here
    SJMP MAIN_LOOP
TIMER1_ISR:             ; Timer 1 interrupt service routine
    ; Your Timer 1 interrupts code here
    RETI                ; Return from interrupt
CAPTURE_ISR:            ; Capture interrupt service routine
    ; Your capture interrupts code here
    RETI                ; Return from interrupt
COMPARE_ISR:            ; Compare interrupt service routine
    ; Your compare interrupt code here
    RETI                ; Return from interrupt
```

In this example, Timer 1 is set in capture and compare mode using the TMOD register. Timer 1 is started by setting the TR1 bit. The main program logic executes in a loop. When Timer 1 overflows, it triggers the Timer 1 interrupt, and the microcontroller jumps to the TIMER1_ISR label to execute the Timer 1 interrupt service routine. Additionally, if you're using the capture or compare features of Timer 1, you can define separate interrupt service routines for capture and compare interrupts (CAPTURE_ISR and COMPARE_ISR labels in the example).