STC15-28-PIN series MCU
Data Sheet

STC MCU Limited
www.STCMCU.com

Update date: 2010-08-06

CONTENTS

Chapter 1 Introduction 4
1.1 Features 4
1.2 Block diagram 5
1.3 PINS Definition 6
1.4 PINS Descriptions 7
1.5 Package Drawings 9
Chapter 2 Clock, Power Managenment, Reset 11
2.1 Clock 11
2.2 Power Management 12
2.2.1 Idle Mode 12
2.2.2 Slow Down Mode 12
2.2.3 Power Down (PD) Mode (Halt Mode) 13
2.3 Reset 18
2.3.1 Reset pin 18
2.3.2 Software RESET 18
2.3.3 Power-On Reset (POR) 18
2.3.4 Watch-Dog-Timer 19
2.3.5 MAX810 power-on-reset delay 23
2.3.6 Low Voltage Detection 23
Chapter 3 Memory Organization 25
3.1 Program Memory 25
3.2 SRAM 26
Chapter 4. Configurable I/O Ports 28
4.1 I/O Port Configurations 28
4.1.1 Quasi-bidirectional I/O 28
4.1.2 Push-pull Output 29
4.1.3 Input-only Mode 29
4.1.4 Open-drain Output 29
4.2 I/O Port Registers 30

Chapter 5 Instruction System.....32
5.1 Special Function Registers..... 32
5.2 Notes on Compatibility to Standard 80C51 MCU..... 36
5.3 Addressing Modes..... 37
5.4 Instruction Set Summary..... 38
5.5 Instruction Definitions..... 43
Chapter 6 Interrupts.....80
6.1 Interrupt Structure..... 81
6.2 Interrupt Register..... 83
6.3 Interrupt Priorities..... 86
6.4 How Interrupts Are Handled..... 87
6.5 External Interrupts..... 88
Chapter 7 Timer/Counter 0 and 1.....89
7.1 Timer/Counter 0 Mode of Operation.....92
7.2 Timer/Counter 1 Mode of Operation..... 96
7.3 Changes of STC15-28-PIN Timers compared with standard 8051 102
7.4 Generic Programmable Clock Output..... 103
Chapter 8 Analog to Digital Converter.....106
8.1 A/D Converter Structure..... 106
8.2 Register for ADC..... 107
8.3 Program using interrupts to demostrate A/D Conversion..... 109
8.4 Program using inquiry to demostrate A/D Conversion.....114
Chapter 9 IAP / EEPROM.....120
9.1 IAP / ISP Control Register..... 120
9.2 IAP/EEPROM Assembly Language Program Introduction..... 123
9.4 EEPROM Demo Program written in Assembly Language..... 125
9.5 EEPROM Demo Program written in C Language..... 129

Chapter 1 Introduction

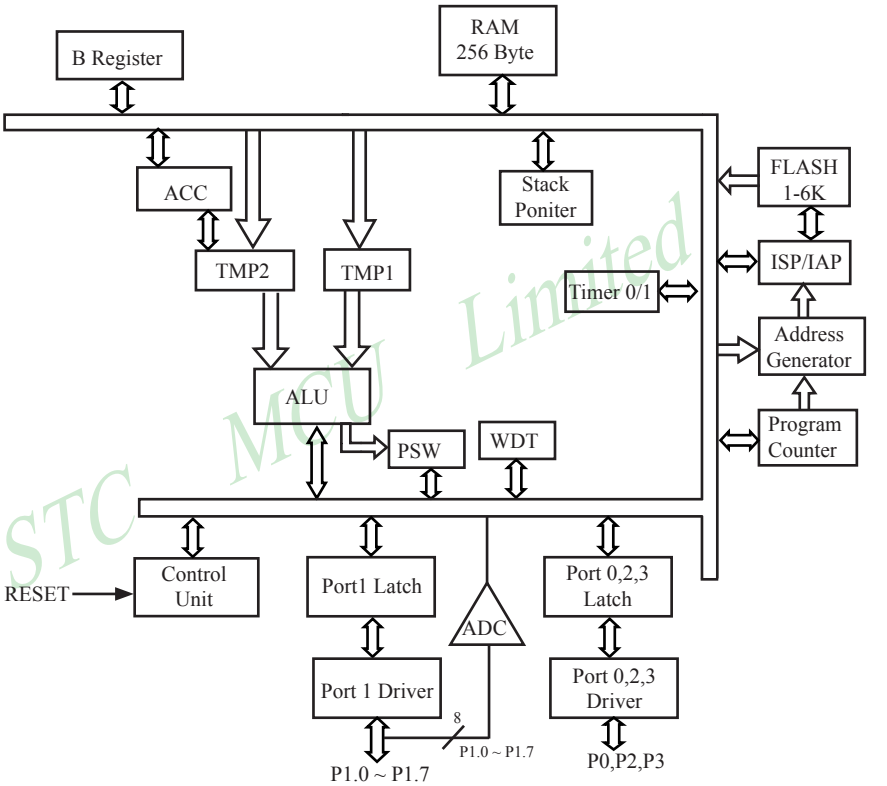
STC15-28-PIN is a single-chip microcontroller based on a high performance 1T architecture 80C51 CPU, which is produced by STC MCU Limited. With the enhanced kernel, STC15-28-PIN executes instructions in 1~6 clock cycles (about 6~7 times the rate of a standard 8051 device), and has a fully compatible instruction set with industrial-standard 80C51 series microcontroller. In-System-Programming (ISP) and In-Application-Programming (IAP) support the users to upgrade the program and data in system. ISP allows the user to download new code without removing the microcontroller from the actual end product; IAP means that the device can write non-valatile data in Flash memory while the application program is running. the STC15-28-PIN has 9 interrupt sources, 10-bit ADC, on-chip RC oscillator and a one-time enabled Watch-Dog Timer.

1.1 Features

- Enhanced 80C51 Central Processing Unit, faster 6~7 times than the rate of a standard 8051
- Operating voltage range: 3.8 ~ 5.5V or 2.4V ~ 3.6V
- Operating frequency range: DC------5.5296MHz(±30%)/11.0592MHz(±30%)/22.1184MHz(±30%) / 33.1776MHz(±30%)
- On-chip 256 bytes RAM and 1K~6K bytes flash
- Code protection for flash memory access
- Two 16-bit timer/counter
- 10-bit ADC associated interrupt
- 9 interrupt sources
- One 15 bits Watch-Dog-Timer with 8-bit pre-scalar (one-time-enabled)
- An internal RC oscillator with adjustable frequency to 5.5296MHz(±30%)/11.0592MHz(±30%)/22.1184MHz(±30%)/33.1776MHz (±30%)
- Three power management modes: idle mode, slow down mode and power-down mode
  Power down mode can be woken-up by external INTx pin and T0/T1 pin
- Support 2-wire serial flash programming interface.(GND/P3.0/P3.1/VCC)
- Programmable clock output Function. T0 output the clock on P3.5, T1 output clock on P3.4.
- Package type: SOP-28,SKDIP-28

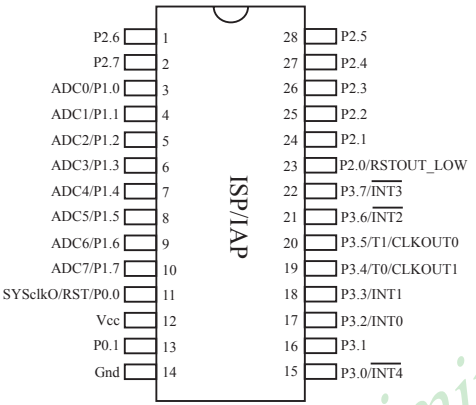
1.2 Block diagram

The CPU kernel of STC15-28-PIN is fully compatible to the standard 8051 microcontroller, maintains all instruction mnemonics and binary compatibility. With some great architecture enhancements, STC15-28-PIN executes the fastest instructions per clock cycle. Improvement of individual programs depends on the actual instructions used.



STC15-28-PIN Block Diagram

1.3 PINS Definition



SOP-28/SKDIP-28

1.4 PINS Descriptions

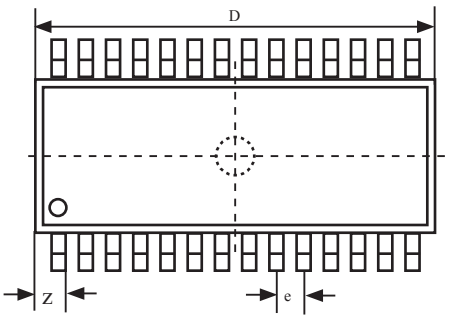
MNEMONIC	Pin number	DESCRIPTION	
P0.0/RST/SYSclkO	11	P0.0	Standard PORT0[0]
		RST	Reset pin;
		SYSclkO	Internal system clock output;
P0.1	13	Standard PORT0[1]	
P1.0/ADC0	3	P1.0	Standard PORT1[0]
		ADC0	ADC input channel-0
P1.1/ADC1	4	P1.1	Standard PORT1[1]
		ADC1	ADC input channel-1
P1.2/ADC2	5	P1.2	Standard PORT1[2]
		ADC2	ADC input channel-2
P1.3/ADC3	6	P1.3	Standard PORT1[3]
		ADC3	ADC input channel-3
P1.4/ADC4	7	P1.4	Standard PORT1[4]
		ADC4	ADC input channel-4
P1.5/ADC5	8	P1.5	Standard PORT1[5]
		ADC5	ADC input channel-5
P1.6/ADC6	9	P1.6	Standard PORT1[6]
		ADC6	ADC input channel-6
P1.7/ADC7	10	P1.7	Standard PORT1[7]
		ADC7	ADC input channel-7
P2.0/ RSTOUT_LOW	23	P2.0	Standard PORT2[0]
		RSTOUT_LOW	After power-up, it will output 0. Change the output register to 1 before making it iuput
P2.1	24	Standard PORT2[1]	
P2.2	25	Standard PORT2[2]	
P2.3	26	Standard PORT2[3]	
P2.4	27	Standard PORT2[4]	
P2.5	28	Standard PORT2[5]	
P2.6	1	Standard PORT2[6]	
P2.7	2	Standard PORT2[7]	
P3.0/$\overline{\text{INT4}}$	15	P3.0	Standard PORT3[0]
		$\overline{\text{INT4}}$	One of external Interrupt sources. The interrupting acts in Negative-Edge only, and with Lease priority, and it can wake up the STC15-28-PIN from power-down mode.
P3.1	16	Standard PORT3[1]	

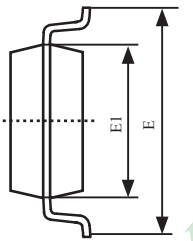
MNEMONIC	Pin number	DESCRIPTION	
P3.2/INT0	17	P3.2	Standard PORT3[2]
		INT0	One of external Interrupt sources. The interrupt acting can be configured to Negative-Edge-Active or On-Change-Active(Negative-Edge-Active and Positive-Edge-Active). A Negative-Edge from INT0 pin will trigger an interrupt if IT0(TCON.0) is set, and both of Negative-Edge and Positive-Edge will trigger an interrupt if IT0(TCON.0) is cleared. Also INT0 can wake up the STC15-28-PIN from power-down mode.
P3.3/INT1	18	P3.3	Standard PORT3[3]
		INT1	One of external Interrupt sources. The interrupt acting can be configured to Negative-Edge-Active or On-Change-Active(Negative-Edge-Active and Positive-Edge-Active). A Negative-Edge from INT1 pin will trigger an interrupt if IT1(TCON.2) is set, and both of Negative-Edge and Positive-Edge will trigger an interrupt if IT1(TCON.2) is cleared. Also INT1 can wake up the STC15-28-PIN from power-down mode.
P3.4/T0/CLKOUT1	19	P3.4	Standard PORT3[4]
		T0	T0 input for Timer 0
		CLKOUT1	Frequency output associated with TIMER-1 overflow rate divided by 2 Set INT_CLKO[1](T1CLKO)=1 to act it.
P3.5/T1/CLKOUT0	20	P3.5	Standard PORT3[5]
		T1	T1 input for Timer 1
		CLKOUT0	Frequency output associated with TIMER-0 overflow rate divided by 2 Set INT_CLKO[0](T0CLKO)=1 to act it.
P3.6/$\overline{INT2}$	21	P3.6	Standard PORT3[6]
		$\overline{INT2}$	One of external Interrupt sources. The interrupting acts in Negative-Edge only, and with Lease priority, and it can wake up the STC15-28-PIN from power-down mode.
P3.7/$\overline{INT3}$	22	P3.7	Standard PORT3[7]
		$\overline{INT3}$	One of external Interrupt sources. The interrupting acts in Negative-Edge only, and with Lease priority, and it can wake up the STC15-28-PIN from power-down mode.
Vcc	12	Power	
Gnd	14	Ground	

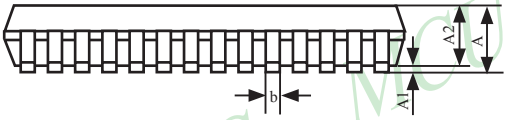
1.5 Package Drawings

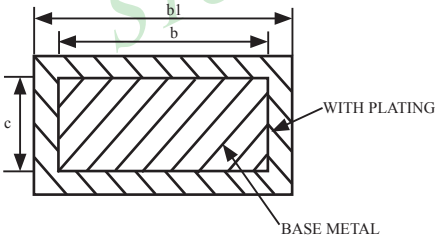
28-Pin Small Outline Package (SOP-28)

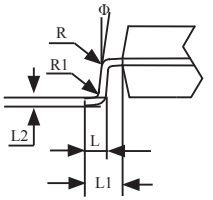
Dimensions in Millimeters







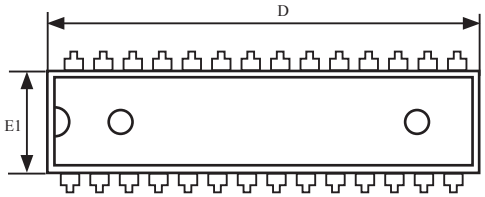


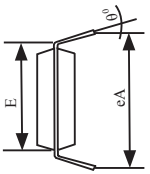


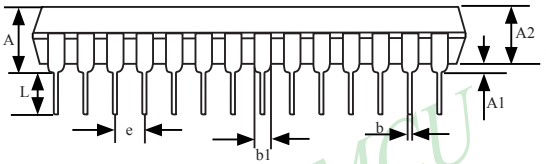
COMMON DIMENSIONS			
(UNITS OF MEASURE = MILLMETER)			
SYMBOL	MIN	NOM	MAX
A	2.465	2.515	2.565
A1	0.100	0.150	0.200
A2	2.100	2.300	2.500
b	0.356	0.406	0.456
b1	0.366	0.426	0.486
c	-	0.254	-
D	17.750	17.950	18.150
E	10.100	10.300	10.500
E1	7.424	7.500	7.624
e		1.27	
L	0.764	0.864	0.964
L1	1.303	1.403	1.503
L2	-	0.274	-
R	-	0.200	-
R1	-	0.300	-
Φ	0^0	-	10^0
z	-	0.745	-

28-Pin Plastic Dual-In-line Package (SKDIP-28)

Dimensions in Inches and Millmeters







COMMON DIMENSIONS (UNITS OF MEASURE = INCH)			
SYMBOL	MIN	NOM	MAX
A	-	-	0.210
A1	0.015	-	-
A2	0.125	0.13	0.135
b	-	0.018	-
b1	-	0.060	-
D	1.385	1.390	1.40
E	-	0.310	-
E1	0.283	0.288	0.293
e	-	0.100	-
L	0.115	0.130	0.150
θ^0	0	7	15
eA	0.330	0.350	0.370

UNIT: INCH

Chapter 2 Clock, Power Managenment, Reset

2.1 Clock

There is only one clock source—Internal RC oscillator available for STC15-28-PIN.
After picking out clocking source, there is another slow-down mechanism available for power-saving purpose.

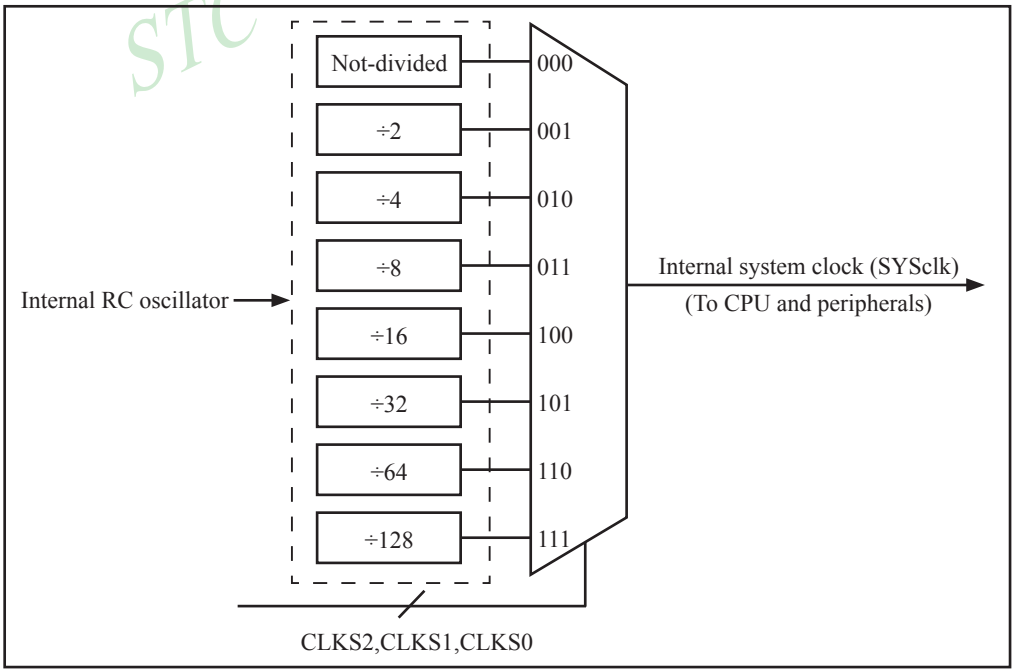
User can slow down the MCU by means of writing a non-zero value to the CLKS[2:0] bits in the CLK_DIV register. This feature is especially useful to save power consumption in idle mode as long as the user changes the CLKS[2:0] to a non-zero value before entering the idle mode.

CLK_DIV register (Clock Divider)

LSB

SFR Name	SFR Address	B7	B6	B5	B4	B3	B2	B1	B0
CLK_DIV	97H	--	--	--	--	--	CLKS2	CLKS1	CLKS0

{CLKS2,CLKS1,CLKS0}
000 := The internal RC oscillator is set as the clock-in not divided (default state)
001 := The internal RC oscillator is set as the clock-in divided by 2
010 := The internal RC oscillator is set as the clock-in divided by 4
011 := The internal RC oscillator is set as the clock-in divided by 8
100 := The internal RC oscillator is set as the clock-in divided by16
101 := The internal RC oscillator is set as the clock-in divided by 32
110 := The internal RC oscillator is set as the clock-in divided by 64
111 := The internal RC oscillator is set as the clock-in divided by 128



Clock Structure

2.2 Power Management

PCON register (Power Control Register)

LSB

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	87H	--	--	LVDF	--	--	--	PD	IDL

LVDF : Low-Voltage Flag. Once low voltage condition is detected (VCC power is lower than LVD voltage), it is set by hardware (and should be cleared by software).
PD : Power-Down bit.
IDL : Idle mode bit.

2.2.1 Idle Mode

An instruction that sets IDL/PCON.0 causes that to be the last instruction executed before going into the idle mode, the internal clock is gated off to the CPU but not to the interrupt, timer, ADC and WDT functions. The CPU status is preserved in its entirety: the RAM, Stack Pointer, Program Counter, Program Status Word, Accumulator, and all other registers maintain their data during Idle. The port pins hold the logical states they had at the time Idle was activated. Idle mode leaves the peripherals running in order to allow them to wake up the CPU when an interrupt is generated. Timer 0, Timer 1 and so on will continue to function during Idle mode.

There are two ways to terminate the idle. Activation of any enabled interrupt will cause IDL/PCON.0 to be cleared by hardware, terminating the idle mode. The interrupt will be serviced, and following RETI, the next instruction to be executed will be the one following the instruction that put the device into idle.
.

The other way to wake-up from idle is to pull RESET high to generate internal hardware reset. Since the clock oscillator is still running, the hardware reset neeeds to be held active for only two machine cycles (24 oscillator periods) to complete the reset.

2.2.2 Slow Down Mode

A divider is designed to slow down the clock source prior to route to all logic circuit. The operating frequency of internal logic circuit can therefore be slowed down dynamically , and then save the power.

2.2.3 Power Down (PD) Mode (Halt Mode)

An instruction that sets PD/PCON.1 cause that to be the last instruction executed before going into the Power-Down mode. In the Power-Down mode, the on-chip oscillator and the Flash memory are stopped in order to minimize power consumption. Only the power-on circuitry will continue to draw power during Power-Down. The contents of on-chip RAM and SFRs are maintained. The power-down mode can be woken-up by RESET pin, external interrupt INT0 ~ INT4, T0 pin and T1 pin. When it is woken-up by RESET, the program will execute from the address 0x0000. Be carefully to keep RESET pin active for at least 10ms in order for a stable clock. If it is woken-up from I/O, the CPU will rework through jumping to related interrupt service routine. Before the CPU rework, the clock is blocked and counted until 32768 in order for denouncing the unstable clock. To use I/O wake-up, interrupt-related registers have to be enabled and programmed accurately before power-down is entered. Pay attention to have at least one “NOP” instruction subsequent to the power-down instruction if I/O wake-up is used. When terminating Power-down by an interrupt, the wake up period is internally timed. At the negative edge on the interrupt pin, Power-Down is exited, the oscillator is restarted, and an internal timer begins counting. The internal clock will be allowed to propagate and the CPU will not resume execution until after the timer has reached internal counter full. After the timeout period, the interrupt service routine will begin. To prevent the interrupt from re-triggering, the interrupt service routine should disable the interrupt before returning. The interrupt pin should be held low until the device has timed out and begun executing. The user should not attempt to enter (or re-enter) the power-down mode for a minimum of 4 us until after one of the following conditions has occured: Start of code execution(after any type of reset), or Exit from power-down mode.

The following example C program demostrates that power-down mode be woken-up by external interrupt .

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU International Limited ---------------------------------------------------------*/
/* --- STC 1T Series MCU wake up Power-Down mode Demo ----------------------------------------*/
/* --- Mobile: (86)13922805190 ---------------------------------------------------------------*/
/* --- Fax: 86-755-82944243 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------*/
/* If you want to use the program or the program referenced in the  */
/* article, please specify in which data and procedures from STC    */
/*---------------------------------------------------------------------------------------------*/

#include <reg51.h>
#include <intrins.h>

sbit        Begin_LED = P1^2;                       //Begin-LED indicator indicates system start-up
unsigned char       Is_Power_Down = 0;              //Set this bit before go into Power-down mode
sbit        Is_Power_Down_LED_INT0          = P1^7; //Power-Down wake-up LED indicator on INT0
sbit        Not_Power_Down_LED_INT0         = P1^6; //Not  Power-Down wake-up LED indicator on INT0
sbit        Is_Power_Down_LED_INT1          = P1^5; //Power-Down wake-up LED indicator on INT1
sbit        Not_Power_Down_LED_INT1         = P1^4; //Not  Power-Down wake-up LED indicator on INT1
sbit        Power_Down_Wakeup_Pin_INT0      = P3^2; //Power-Down wake-up pin on INT0
sbit        Power_Down_Wakeup_Pin_INT1      = P3^3; //Power-Down wake-up pin on INT1
sbit        Normal_Work_Flashing_LED        = P1^3; //Normal work LED indicator

void  Normal_Work_Flashing (void);
void  INT_System_init (void);
void  INT0_Routine (void);
void  INT1_Routine (void);
```

```
void main (void)
{
        unsigned char       j = 0;
        unsigned char       wakeup_counter = 0;
                                        //clear interrupt wakeup counter variable wakeup_counter
        Begin_LED = 0;                  //system start-up LED
        INT_System_init ( );            //Interrupt system initialization
        while(1)
        {
                P2 = wakeup_counter;
                wakeup_counter++;
                for(j=0; j<2; j++)
                {
                        Normal_Work_Flashing( );    //System normal work
                }
                Is_Power_Down = 1;              //Set this bit before go into Power-down mode
                PCON    = 0x02;         //after this instruction, MCU will be in power-down mode
                                        //external clock stop
                _nop_( );
                _nop_( );
                _nop_( );
                _nop_( );
        }
}
void  INT_System_init (void)
{
        IT0     = 0;                    /* External interrupt 0, low electrical level triggered */
//      IT0     = 1;                    /* External interrupt 0, negative edge triggered */
        EX0     = 1;                    /* Enable external interrupt 0
        IT1     = 0;                    /* External interrupt 1, low electrical level triggered */
//      IT1     = 1;                    /* External interrupt 1, negative edge triggered */
        EX1     = 1;                    /* Enable external interrupt 1
        EA      = 1;                    /* Set Global Enable bit
}
void  INT0_Routine (void) interrupt 0
{
        if (Is_Power_Down)
        {
                //Is_Power_Down ==1;        /* Power-Down wakeup on INT0 */
                Is_Power_Down = 0;
                Is_Power_Down_LED_INT0 = 0;
                                /*open external interrupt 0 Power-Down wake-up LED indicator */
                while (Power_Down_Wakeup_Pin_INT0 == 0)
                {
                        /* wait higher */
                }
                Is_Power_Down_LED_INT0 = 1;
                                /* close external interrupt 0 Power-Down wake-up LED indicator */
        }
```

```
        else
        {
                Not_Power_Down_LED_INT0 = 0;    /* open external interrupt 0 normal work LED */
                while (Power_Down_Wakeup_Pin_INT0 ==0)
                {
                        /* wait higher */
                }
                Not_Power_Down_LED_INT0 = 1;    /* close external interrupt 0 normal work LED */
        }
}

void  INT1_Routine (void)   interrupt 2
{
        if (Is_Power_Down)
        {
                //Is_Power_Down ==1;        /* Power-Down wakeup on INT1 */
                Is_Power_Down = 0;
                Is_Power_Down_LED_INT1= 0;
                                /*open external interrupt 1 Power-Down wake-up LED indicator */
                while (Power_Down_Wakeup_Pin_INT1 == 0)
                {
                        /* wait higher */
                }
                Is_Power_Down_LED_INT1 = 1;
                                /* close external interrupt 1 Power-Down wake-up LED indicator */
        }
        else
        {
                Not_Power_Down_LED_INT1 = 0;    /* open external interrupt 1 normal work LED */
                while (Power_Down_Wakeup_Pin_INT1 ==0)
                {
                        /* wait higher */
                }
                Not_Power_Down_LED_INT1 = 1;    /* close external interrupt 1 normal work LED */
        }
}

void  delay (void)
{
        unsigned int        j = 0x00;
        unsigned  int       k = 0x00;
        for (k=0; k<2; ++k)
        {
                for (j=0; j<=30000; ++j)
                {
                        _nop_( );
                        _nop_( );
                        _nop_( );
                        _nop_( );
```

```
                    _nop_( );
                    _nop_( );
                    _nop_( );
                    _nop_( );
                }
        }
}

void Normal_Work_Flashing (void)
{
        Normal_Work_Flashing_LED = 0;
        delay ( );
        Normal_Work_Flashing_LED = 1;
        delay ( );
}
```

The following program also demostrates that power-down mode or idle mode be woken-up by external interrupt, but is written in assembly language rather than C languge.

```
;*************************************************************
;Wake Up Idle and Wake Up Power Down
;*************************************************************
                ORG     0000H
                AJMP    MAIN
                ORG     0003H
int0_interrupt:
                CLR     P1.7            ;open P1.7 LED indicator
                ACALL   delay           ;delay in order to observe
                CLR     EA              ;clear global enable bit, stop all interrupts
                RETI

                ORG     0013H
int1_interrupt:
                CLR     P1.6            ;open P1.6 LED indicator
                ACALL   delay           ;;delay in order to observe
                CLR     EA              ;clear global enable bit, stop all interrupts
                RETI

                ORG     0100H
delay:
                CLR     A
                MOV     R0,     A
                MOV     R1,     A
                MOV     R2,     #02
delay_loop:
                DJNZ    R0,     delay_loop
                DJNZ    R1,     delay_loop
                DJNZ    R2,     delay_loop
                RET
```

```
main:
                MOV     R3,     #0              ;P1 LED increment mode changed
                                                ;start to run program
main_loop:
                MOV     A,      R3
                CPL     A
                MOV     P1,     A
                ACALL   delay
                INC     R3
                MOV     A,      R3
                SUBB    A,      #18H
                JC      main_loop
                MOV     P1,     #0FFH           ;close all LED, MCU go into power-down mode
                CLR     IT0                     ;low electrical level trigger external interrupt 0
        ;       SETB    IT0                     ;negative edge trigger external interrupt 0
                SETB    EX0                     ;enable external interrupt 0
                CLR     IT1                     ;low electrical level trigger external interrupt 1
        ;       SETB    IT1                     ;negative edge trigger external interrupt 1
                SETB    EX1                     ;enable external interrupt 1
                SETB    EA                      ;set the global enable
                                                ;if don't so, power-down mode cannot be wake up

;MCU will go into idle mode or power-down mode after the following instructions
                MOV     PCON,   #00000010B      ;Set PD bit, power-down mode (PD = PCON.1)
        ;       NOP
        ;       NOP
        ;       NOP
        ;       MOV     PCON,   #00000001B      ;Set IDL bit, idle mode (IDL = PCON.0)
                MOV     P1,     #0DFH           ;1101,1111
                NOP
                NOP
                NOP
WAIT1:
                SJMP    WAIT1                   ;dynamically stop
                END
```

2.3 Reset

In STC15-28-PIN, there are 6 sources to generate internal reset. They are RESET (P0.0) pin, software reset, On-chip power-on-reset,Watch-Dog-Timer,On-chip MAX810 POR timing delay and low-voltage detection.

Those following conditions will induce reset.

- (User-Invoked) Reset pin acting
- (User-Invoked) Software Reset via SWRST (IAP_CONTR.5)
- (System-Invoked) Basic Power-Up latency (8192 clocks)
- (System-Invoked) Watch-Dog-Timer overflow
- (System-Invoked) MAX810-like Power-Up latency (~45mS)
- (System-Invoked) Low-Voltage detector acting
- (System-Invoked) Invalid MOVC Address
- (System-Invoked) Invalid IAP Address
- (System-Invoked) Invalid Program Address

2.3.1 Reset pin

The P0.0 pin, if configured as RESET pin function, which is the input to Schmitt Trigger, is input pin for chip reset. A level change of RESET pin have to keep at least 24 cycles plus 10us in order for CPU internal sampling use.

2.3.2 Software RESET

Writing an “1” to SWRST bit in IAP_CONTR register will generate a internal reset.

IAP_CONTR: ISP/IAP Control Register

SFR Name	SFR Address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	--	WT2	WT1	WT0

SWBS : software boot selection control bit
0 : Boot from user-code after reset
1 : Boot from ISP monitor code after reset
SWRST : software reset trigger control.
0 : No operation
1 : Generate software system reset. It will be cleared by hardware automatically

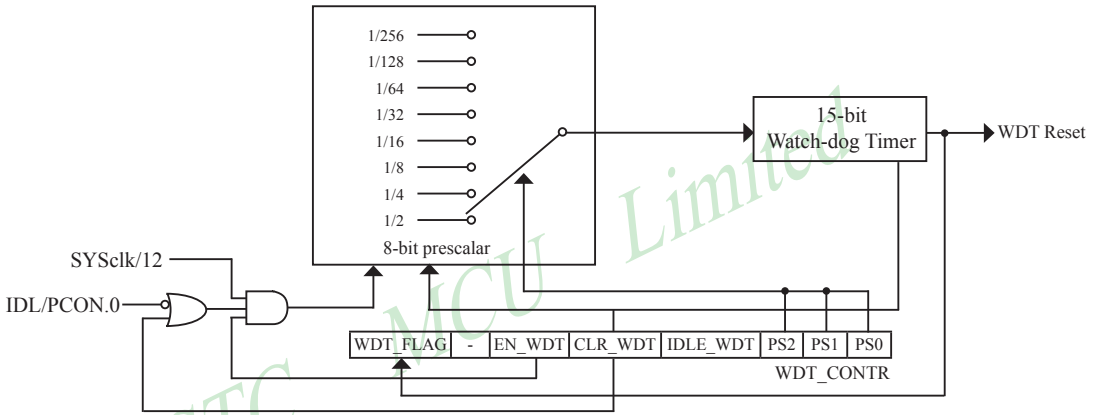
2.3.3 Power-On Reset (POR)

When VCC drops below the detection threshold of POR circuit, all of the logic circuits are reset.

When VCC goes back up again, an internal reset is released automatically after a delay of 32768 clocks.

2.3.4 Watch-Dog-Timer

The watch dog timer in STC15-28-PIN consists of an 8-bit pre-scaler timer and an 15-bit timer. The timer is one-time enabled by setting EN_WDT(WDT_CONTR.5). Clearing EN_WDT can stop WDT counting. When the WDT is enabled, software should always reset the timer by writing 1 to CLR_WDT bit before the WDT overflows. If STC15-28-PIN out of control by any disturbance, that means the CPU can not run the software normally, then WDT may miss the "writting 1 to CLR_WDT" and overflow will come. An overflow of Watch-Dog-Timer will generate a internal reset.



WDT Structure

WDT_CONTR: Watch-Dog-Timer Control Register

LSB

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	0C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDLE_WDT	PS2	PS1	PS0

WDT_FLAG : WDT reset flag.
0 : This bit should be cleared by software.
1 : When WDT overflows, this bit is set by hardware to indicate a WDT reset happened.
EN_WDT : Enable WDT bit. When set, WDT is started.
CLR_WDT : WDT clear bit. When set, WDT will recount. Hardware will automatically clear this bit.
IDLE_WDT : WDT IDLE mode bit. When set, WDT is enabled in IDLE mode. When clear, WDT is disabled in IDLE.

PS2, PS1, PS0 : WDT Pre-scale value set bit.
Pre-scale value of Watchdog timer is shown as the bellowed table :

PS2	PS1	PS0	Pre-scale	WDT overflow Time @20MHz
0	0	0	2	39.3 mS
0	0	1	4	78.6 mS
0	1	0	8	157.3 mS
0	1	1	16	314.6 mS
1	0	0	32	629.1 mS
1	0	1	64	1.25 S
1	1	0	128	2.5 S
1	1	1	256	5 S

The WDT overflow time is determined by the following equation:

WDT overflow time = (12 × Pre-scale × 32768) / SYSclk

The SYSclk is 20MHz in the table above.
If SYSclk is 12MHz, The WDT overflow time is :

WDT overflow time = (12 × Pre-scale × 32768) / 12000000 = Pre-scale× 393216 / 12000000

WDT overflow time is shown as the bellowed table when SYSclk is 12MHz:

PS2	PS1	PS0	Pre-scale	WDT overflow Time @12MHz
0	0	0	2	65.5 mS
0	0	1	4	131.0 mS
0	1	0	8	262.1 mS
0	1	1	16	524.2 mS
1	0	0	32	1.0485 S
1	0	1	64	2.0971 S
1	1	0	128	4.1943 S
1	1	1	256	8.3886 S

WDT overflow time is shown as the bellowed table when SYSclk is 11.0592MHz:

PS2	PS1	PS0	Pre-scale	WDT overflow Time @11.0592MHz
0	0	0	2	71.1 mS
0	0	1	4	142.2 mS
0	1	0	8	284.4 mS
0	1	1	16	568.8 mS
1	0	0	32	1.1377 S
1	0	1	64	2.2755 S
1	1	0	128	4.5511 S
1	1	1	256	9.1022 S

The following example is a assembly language program that demostrates STC 1T Series MCU WDT.

```
;/*---------------------------------------------------------------*/
;/* --- STC MCU International Limited ----------------------------*/
;/* --- STC 1T Series MCU WDT Demo -------------------------------*/
;/* --- Mobile: (86)13922805190 ----------------------------------*/
;/* --- Fax: 86-755-82944243 -------------------------------------*/
;/* --- Tel: 86-755-82948412 -------------------------------------*/
;/* --- Web: www.STCMCU.com --------------------------------------*/
;/* If you want to use the program or the program referenced in the  */
;/* article, please specify in which data and procedures from STC    */
;/*---------------------------------------------------------------*/
; WDT overflow time = (12 × Pre-scale × 32768) / SYSclk
WDT_CONTR               EQU     0C1H            ;WDT address
WDT_TIME_LED            EQU     P1.5            ;WDT overflow time LED on P1.5
                                                ;The WDT overflow time may be measured by the LED light time
WDT_FLAG_LED            EQU     P1.7
                                                ;WDT overflow reset flag LED indicator on P1.7
Last_WDT_Time_LED_Status        EQU     00H
                        ;bit variable used to save the last stauts of WDT overflow time LED indicator

;WDT reset time , the SYSclk is 18.432MHz
;Pre_scale_Word  EQU     00111100B       ;open WDT, Pre-scale value is 32, WDT overflow time=0.68S
;Pre_scale_Word  EQU     00111101B       ;open WDT, Pre-scale value is 64, WDT overflow time=1.36S
;Pre_scale_Word  EQU     00111110B       ;open WDT, Pre-scale value is 128, WDT overflow time=2.72S
;Pre_scale_Word  EQU     00111111B       ;open WDT, Pre-scale value is 256, WDT overflow time=5.44S

                ORG     0000H
                AJMP    MAIN
                ORG     0100H
MAIN:
                MOV     A,      WDT_CONTR               ;detection if WDT reset
                ANL     A,      #10000000B
                JNZ     WDT_Reset
                        ;WDT_CONTR.7=1, WDT reset, jump WDT reset subroutine
                        ;WDT_CONTR.7=0, Power-On reset, cold start-up, the content of RAM is  random
                SETB    Last_WDT_Time_LED_Status        ;Power-On reset
                CLR     WDT_TIME_LED            ;Power-On reset,open WDT overflow time LED
                MOV     WDT_CONTR,      #Pre_scale_Word         ;open WDT
```

```
WAIT1:
        SJMP    WAIT1                   ;wait WDT overflow reset
    ;WDT_CONTR.7=1, WDT reset, hot strart-up, the content of RAM is constant and just like before reset
WDT_Reset:
        CLR     WDT_FLAG_LED
                                        ;WDT reset,open WDT overflow reset flag LED indicator
        JB      Last_WDT_Time_LED_Status,       Power_Off_WDT_TIME_LED
        ;when set Last_WDT_Time_LED_Status, close the  corresponding LED indicator
        ;clear, open the  corresponding LED indicator
        ;set WDT_TIME_LED according to the last status of WDT overflow time LED indicator
        CLR     WDT_TIME_LED            ;close the WDT overflow time LED indicator
        CPL     Last_WDT_Time_LED_Statu
                                        ;reverse the last status of WDT overflow time LED indicator

WAIT2:
        SJMP    WAIT2                           ;wait WDT overflow reset
Power_Off_WDT_TIME_LED:
        SETB    WDT_TIME_LED            ;close the WDT overflow time LED indicator
        CPL     Last_WDT_Time_LED_Status
                                        ;reverse the last status of WDT overflow time LED indicator
WAIT3:
        SJMP    WAIT3                           ;wait WDT overflow reset
        END
```

2.3.5 MAX810 power-on-reset delay

There is another on-chip POR delay circuit is integrated on STC15-28-PIN. This circuit is MAX810—sepcial reset circuit and is controlled by configuring flash Option Register. Very long POR delay time – around 45ms will be generated by this circuit once it is enabled.

2.3.6 Low Voltage Detection

Besides the POR voltage, there is a higher threshold voltage: the Low Voltage Detection (LVD) voltag for STC15-28-PIN. When the VCC power drops down to the LVD voltage, the Low voltage Flag, LVD bit (PCON.5), will be set by hardware. (Note that during power-up, this flag will also be set, and the user should clear it by software for the following Low Voltage detecting.) This flag can also generate an interrupt if bit ELVD (IE.6) is set to 1.

The following table lists all the low voltage detection threshold voltages for STC-15-28-PIN operating different degree

Set Value	-40 degree		25 degree		85 degree	
	5V device	3V device	5V device	3V device	5V device	3V device
000	3.21	1.90	3.14	1.89	3.09	1.89
001	3.36	2.01	3.28	2.00	3.23	2.00
010	3.51	2.14	3.43	2.12	3.38	2.12
011	3.69	2.29	3.61	2.26	3.56	2.26
100	3.90	2.44	3.82	2.42	3.77	2.43
101	4.14	2.63	4.05	2.61	4.00	2.61
110	4.41	2.85	4.32	2.82	4.27	2.83
111	4.74	3.11	4.64	3.08	4.60	3.09

There is 2us noise filter embedded inter the LVD against noise impact.

Some SFRs related to Low voltage detection as shown below.

PCON register (Power Control Register)

LSB

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	87H	--	--	LVDF	--	--	--	PD	IDL

LVDF : Low-Voltage Flag. Once low voltage condition is detected (VCC power is lower than LVD voltage), it is set by hardware (and should be cleared by software).

IE: Interrupt Enable Rsgister

(MSB)							(LSB)
EA	ELVD	EADC	-	ET1	EX1	ET0	EX0

Enable Bit = 1 enables the interrupt .
Enable Bit = 0 disables it .

EA (IE.7): disables all interrupts. if EA = 0,no interrupt will be acknowledged. if EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

ELVD (IE.6): Low volatge detection interrupt enable bit.

IP: Interrupt Priority Register

(MSB)							(LSB)
--	PLVD	PADC	--	PT1	PX1	PT0	PX0

Priority bit = 1 assigns high priority .
Priority bit = 0 assigns low priority.

PLVD (IP.6): Low voltage detection interrupt priority.

Chapter 3 Memory Organization

The STC15-28-PIN series MCU has separate address space for Program Memory and Data Memory. The logical separation of program and data memory allows the data memory to be accessed by 8-bit addresses, which can be quickly stored and manipulated by the CPU.

Program memory (ROM) can only be read, not written to. In the STC15-28-PIN series, all the program memory are on-chip Flash memory, and without the capability of accessing external program memory because of no External Access Enable (/EA) and Program Store Enable (/PSEN) signals designed.

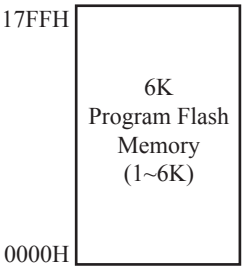
Data memory occupies a separate address space from program memory. In the STC15-28-PIN series, there are 256 bytes of internal scratch-pad RAM(SRAM).

3.1 Program Memory

Program memory is the memory which stores the program codes for the CPU to execute. There is 8K-bytes of flash memory embedded for program and data storage in STC15-28-PIN. The design allows users to configure it as like there are three individual partition banks inside. They are called AP(application program) region, IAP (In-Application-Program) region and ISP (In-System-Program) boot region. AP region is the space that user program is resided. IAP(In-Application-Program) region is the nonvolatile data storage space that may be used to save important parameters by AP program. In other words, the IAP capability of STC15-28-PIN provides the user to read/write the user-defined on-chip data flash region to save the needing in use of external EEPROM device. ISP boot region is the space that allows a specific program we calls “ISP program” is resided. Inside the ISP region, the user can also enable read/write access to a small memory space to store parameters for specific purposes. Generally, the purpose of ISP program is to fulfill AP program upgrade without the need to remove the device from system. STC15-28PIN hardware catches the configuration information since power-up duration and performs out-of-space hardware-protection depending on pre-determined criteria. The criteria is AP region can be accessed by ISP program only, IAP region can be accessed by ISP program and AP program, and ISP region is prohibited access from AP program and ISP program itself. But if the “ISP data flash is enabled”, ISP program can read/write this space. When wrong settings on ISP-IAP SFRs are done, The “out-of-space” happens and STC15-28-PIN follows the criteria above, ignore the trigger command.

After reset, the CPU begins execution from the location 0000H of Program Memory, where should be the starting of the user’s application code. To service the interrupts, the interrupt service locations (called interrupt vectors) should be located in the program memory. Each interrupt is assigned a fixed location in the program memory. The interrupt causes the CPU to jump to that location, where it commences execution of the service routine. External Interrupt 0, for example, is assigned to location 0003H. If External Interrupt 0 is going to be used, its service routine must begin at location 0003H. If the interrupt is not going to be used, its service location is available as general purpose program memory.

The interrupt service locations are spaced at an interval of 8 bytes: 0003H for External Interrupt 0, 000BH for Timer 0, 0013H for External Interrupt 1, 001BH for Timer 1, etc. If an interrupt service routine is short enough (as is often the case in control applications), it can reside entirely within that 8-byte interval. Longer service routines can use a jump instruction to skip over subsequent interrupt locations, if other interrupts are in use.

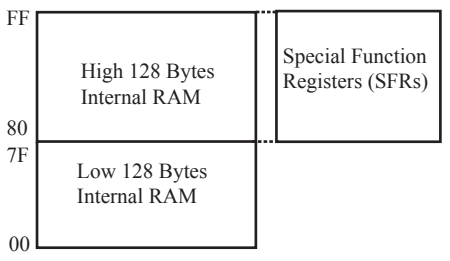


STC15-18-PIN Program Memory

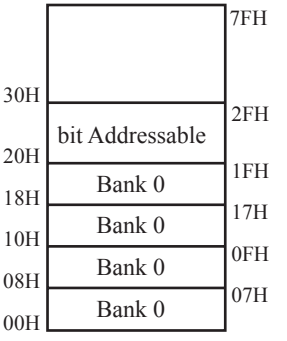
3.2 SRAM

Just the same as the conventional 8051 micro-controller, there are 256 bytes of SRAM data memory plus 128 bytes of SFR space available on the STC15-28-PIN. The lower 128 bytes of data memory may be accessed through both direct and indirect addressing. The upper 128 bytes of data memory and the 128 bytes of SFR space share the same address space. The upper 128 bytes of data memory may only be accessed using indirect addressing. The 128 bytes of SFR can only be accessed through direct addressing. The lowest 32 bytes of data memory are grouped into 4 banks of 8 registers each. Program instructions call out these registers as R0 through R7. The RS0 and RS1 bits in PSW register select which register bank is in use. Instructions using register addressing will only access the currently specified bank. This allows more efficient use of code space, since register instructions are shorter than instructions that use direct addressing. The next 16 bytes (20H~2FH) above the register banks form a block of bit-addressable memory space. The 80C51 instruction set includes a wide selection of single-bit instructions, and the 128 bits in this area can be directly addressed by these instructions. The bit addresses in this area are 00H through 7FH.

All of the bytes in the Lower 128 can be accessed by either direct or indirect addressing while the Upper 128 can only be accessed by indirect addressing. SFRs include the Port latches, timers, peripheral controls, etc. These registers can only be accessed by direct addressing. Sixteen addresses in SFR space are both byte- and bit-addressable. The bit-addressable SFRs are those whose address ends in 0H or 8H.



On-chip Scratch-Pad RAM



Lower 128 Bytes of internal SRAM

PSW register

LSB

bit	B7	B6	B5	B4	B3	B2	B1	B0
name	CY	AC	F0	RS1	RS0	OV	-	P

CY : Carry flag.
AC : Auxilliary Carry Flag.(For BCD operations)
F0 : Flag 0.(Available to the user for general purposes)
RS1: Register bank select control bit 1.
RS0: Register bank select control bit 0.
OV : Overflow flag.
B1 : Reserved
P : Parity flag.

Chapter 4. Configurable I/O Ports

4.1 I/O Port Configurations

STC15-28-PIN have 26 configurable I/O ports: P0.0~P0.1, P1.0~P1.7, P2.0~P2.7, P3.0~P3.7. Port 0 is an 2-bit bi-directional I/O port with pull-up resistance. Port1 is general-purposed I/O with weak pull-up resistance inside. When 1s are written into Port1, the strong output driving CMOS only turn-on two period and then the weak pull-up resistance keep the port high. Port2 is an 8-bit bi-directional I/O port with pull-up resistance. Port3 is general-purposed I/O with weak pull-up resistance inside. When 1s are written into Port1, the strong output driving CMOS only turn-on two period and then the weak pull-up resistance keep the port high. Port3 also serves the functions of various special features.

All port pins on STC15-28-PIN may be independently configured to one of four modes : quasi-bidirectional (standard 8051 port output), push-pull output, input-only or open-drain output .All port pins default to quasi-bidirectional after reset. Each one has a Schmitt-triggered input for improved input noise rejection.

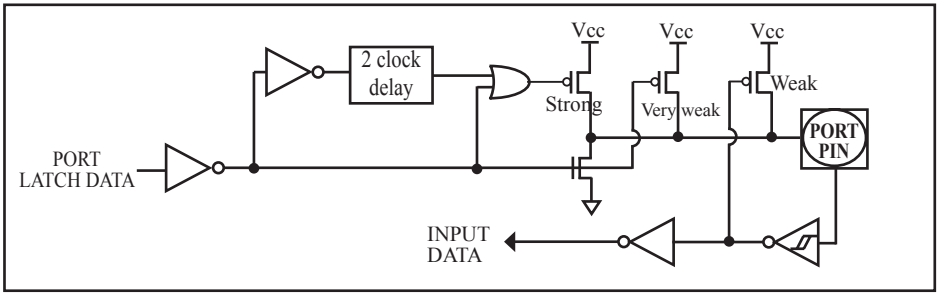
4.1.1 Quasi-bidirectional I/O

Port pins in quasi-bidirectional output mode function similar to the standard 8051 port pins. A quasi-bidirectional port can be used as an input and output without the need to reconfigure the port. This is possible because when the port outputs a logic high, it is weakly driven, allowing an external device to pull the pin low. When the pin outputs low, it is driven strongly and able to sink a large current. There are three pull-up transistors in the quasi-bidirectional output that serve different purposes.

One of these pull-ups, called the “very weak” pull-up, is turned on whenever the port register for the pin contains a logic “1”. This very weak pull-up sources a very small current that will pull the pin high if it is left floating.

A second pull-up, called the “weak” pull-up, is turned on when the port register for the pin contains a logic “1” and the pin itself is also at a logic “1” level. This pull-up provides the primary source current for a quasi-bidirectional pin that is outputting a 1. If this pin is pulled low by the external device, this weak pull-up turns off, and only the very weak pull-up remains on. In order to pull the pin low under these conditions, the external device has to sink enough current to over-power the weak pull-up and pull the port pin below its input threshold voltage.

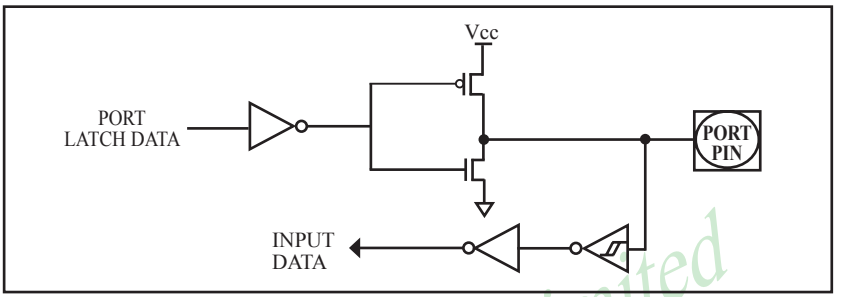
The third pull-up is referred to as the “strong” pull-up. This pull-up is used to speed up low-to-high transitions on a quasi-bidirectional port pin when the port register changes from a logic “0” to a logic “1”. When this occurs, the strong pull-up turns on for two CPU clocks, quickly pulling the port pin high.



Quasi-bidirectional output

4.1.2 Push-pull Output

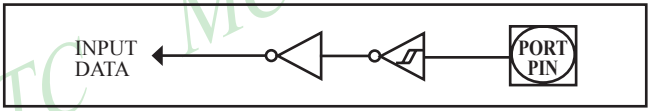
The push-pull output configuration has the same pull-down structure as both the open-drain and the quasi-bidirectional output modes, but provides a continuous strong pull-up when the port register conatins a logic “1”. The push-pull mode may be used when more source current is needed from a port output. In addition, input path of the port pin in this configuration is also the same as quasi-bidirectional mode.



Push-pull output

4.1.3 Input-only Mode

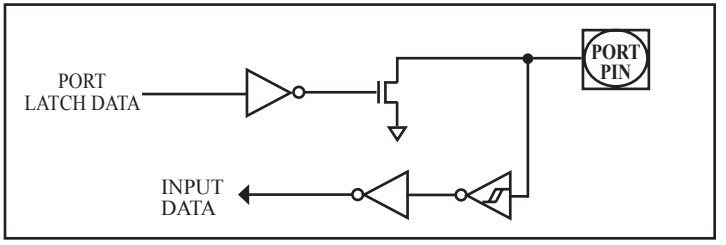
The input-only configuration is a Schmitt-triggered input without any pull-up resistors on the pin.



Input-only Mode

4.1.4 Open-drain Output

The open-drain output configuration turns off all pull-ups and only drives the pull-down transistor of the port pin when the port register contains a logic “0”. To use this configuration in application, a port pin must have an external pull-up, typically tied to VCC. The input path of the port pin in this configuration is the same as quasi-bidirection mode.



Open-drain output

4.2 I/O Port Registers

All port pins on STC15-28-PIN may be independently configured by software to one of four types on a bit-by-bit basis,as shown in next Table.Two mode registers for each port select the output mode for each port pin.

Table: Configuration of I/O port mode.

PxM1.n	PxM0.n	Port Mode
0	0	Quasi-bidirectional
0	1	Push-Pull output
1	0	Input Only (High-impedance)
1	1	Open-Drain Output

where x = 0 ~ 4 (port number), and n = 0 ~7 (port pin).

P0 register

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	80H	-	-	-	-	-	-	P0.1	P0.0

P0 register could be bit-addressable. And P0.1~P0.0 coulde be set/cleared by CPU.

P0M1 register

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	93H	-	-	-	-	-	-	P0M1.1	P0M1.0

P0M0 register

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	94H	-	-	--	-	-	-	P0M0.1	P0M0.0

P1 register

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	90H	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0

P1 register could be bit-addressable and set/cleared by CPU. And P1.7~P1.0 coulde be set/cleared by CPU.

P1M1 register

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	91H	P1M1.7	P1M1.6	P1M1.5	P1M1.4	P1M1.3	P1M1.2	P1M1.1	P1M1.0

P1M0 register

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	92H	P1M0.7	P1M0.6	P1M0.5	P1M0.4	P1M0.3	P1M0.2	P1M0.1	P1M0.0

P2 register

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	A0H	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0

P1 register could be bit-addressable and set/cleared by CPU. And P1.7~P1.0 coulde be set/cleared by CPU.

P2M1 register

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	95H	P2M1.7	P2M1.6	P2M1.5	P2M1.4	P2M1.3	P2M1.2	P2M1.1	P2M1.0

P2M0 register

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	96H	P2M0.7	P2M0.6	P2M0.5	P2M0.4	P2M0.3	P2M0.2	P2M0.1	P2M0.0

P3 register

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	B0H	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0

P3 register could be bit-addressable and set/cleared by CPU. And P3.7~P3.0 coulde be set/cleared by CPU.

P3M1 register

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	B1H	P3M1.7	P3M1.6	P3M1.5	P3M1.4	P3M1.3	P3M1.2	P3M1.1	P3M1.0

P3M0 register

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	B2H	P3M0.7	P3M0.6	P3M0.5	P3M0.4	P3M0.3	P3M0.2	P3M0.1	P3M0.0

Chapter 5 Instruction System

5.1 Special Function Registers

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
0F8H									0FFH
0F0H	B 0000,0000								0F7H
0E8H									0EFH
0E0H	ACC 0000,0000								0E7H
0D8H									0DFH
0D0H	PSW 0000,00x0								0D7H
0C8H									0CFH
0C0H		WDT_CONR 0x00,0000	IAP_DATA 1111,1111	IAP_ADDRH 0000,0000	IAP_ADDRL 0000,0000	IAP_CMD xxxx,xx00	IAP_TRIG xxxx,xxxx	IAP_CONTR 0000,0000	0C7H
0B8H	IP x0x0,0000			IRC_CLKO 0xxx,0xxxx	ADC_CONTR 0000,0000	ADC_RES 0000,0000	ADC_RESL 0000,0000		0BFH
0B0H	P3 1111,1111	P3M1 0000,0000	P3M0 0000,0000						0B7H
0A8H	IE 000x,0000								0AFH
0A0H	P2 1111,1111							Don't use	0A7H
098H						P1ASF 0000,0000	Don't use	Don't use	09FH
090H	P1 1111,1111	P1M1 0000,0000	P1M0 0000,0000	P0M1 0000,0000	P0M0 0000,0000	P2M1 0000,0000	P2M0 0000,0000	CLK_DIV xxxx,x000	097H
088H	TCON 0000,0000	TMOD 0000,0000	TL0 0000,0000	TL1 0000,0000	TH0 0000,0000	TH1 0000,0000	AUXR 00xx,xxxx	INT_CLKO x000,xx00	08FH
080H	P0 1111,1111	SP 0000,0111	DPL 0000,0000	DPH 0000,0000				PCON xx1x,xx00	087H
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

Bit Addressable

Non Bit Addressable

Symbol	Description	Address	Bit Address and Symbol MSB LSB								Value after Power-on or Reset
P0	Port 0	80H	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	1111 1111B
SP	Stack Pointer	81H									0000 0111B
DPTR DPL	Data Pointer Low	82H									0000 0000B
DPTR DPH	Data Pointer High	83H									0000 0000B
PCON	Power Control	87H	-	-	LVDF	-	-	-	PD	IDL	xx1x xx00B
TCON	Timer Control	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000 0000B
TMOD	Timer Mode	89H	GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0	0000 0000B
TL0	Timer Low 0	8AH									0000 0000B
TL1	Timer Low 1	8BH									0000 0000B
TH0	Timer High 0	8CH									0000 0000B
TH1	Timer High 1	8DH									0000 0000B
AUXR	Auxiliary register	8EH	T0x12	T1x12	-	-	-	-	-	-	00xx xxxxB
INT_CLKO	External interrupt Enable and Clock Output register	8FH	-	EX4	EX3	EX2	-	-	T1CLKO	T0CLKO	x000 xx00B
P1	Port 1	90H	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	1111 1111B
P1M1	P1 configuration 1	91H									0000 0000B
P1M0	P1 configuration 0	92H									0000 0000B
P0M1	P0 configuration 1	93H									0000 0000B
P0M0	P0 configuration 0	94H									0000 0000B
P2M1	P2 configuration 1	95H									0000 0000B
P2M0	P2 configuration 0	96H									0000 0000B
CLK_DIV	Clock Divder	97h	-	-	-	-	-	CLKS2	CLKS1	CLKS0	xxxx x000B
P1ASF	P1 Analog Function Configure register	9DH	P17ASF	P16ASF	P15ASF	P14ASF	P13ASF	P12ASF	P11ASF	P10ASF	0000 0000B
P2	Port 2	A0H	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	1111 1111B
IE	Interrupt Enable	A8H	EA	ELVD	EADC	-	ET1	EX1	ET0	EX0	000x 0000B
P3	Port 3	B0H	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0	1111 1111B
P3M1	P3 configuration 1	B1H									0000 0000B
P3M0	P3 configuration 0	B2H									0000 0000B
IP	Interrupt Priority Low	B8H	-	PLVD	PADC	-	PT1	PX1	PT0	PX0	x00x 0000B
IRC_CLKO	Internal RC clock output	BBH	EN_IRCO	--	--	--	DIVIRCO	--	--	--	0xxx,0xxxB

Symbol	Description	Address	Bit Address and Symbol MSB LSB								Value after Power-on or Reset
ADC_CONTR	ADC Control Register	BCH	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0	0000 0000B
ADC_RES	ADC Result high	BDH									0000 0000B
ADC_RESL	ADC Result low	BEH									0000 0000B
WDT_CONTR	Watch-Dog-Timer Control Register	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDLE_WDT	PS2	PS1	PS0	0x00 0000B
IAP_DATA	ISP/IAP Flash Data Register	C2H									1111 1111B
IAP_ADDRH	ISP/IAP Flash Address High	C3H									0000 0000B
IAP_ADDRL	ISP/IAP Flash Address Low	C4H									0000 0000B
IAP_CMD	ISP/IAP Flash Command Register	C5H	-	-	-	-	-	-	MS1	MS0	xxxx xx00B
IAP_TRIG	ISP/IAP Flash Command Trigger	C6H									xxxx xxxxB
IAP_CONTR	ISP/IAP Control Register	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	WT2	WT1	WT0	0000 x000B
PSW	Program Status Word	D0H	CY	AC	F0	RS1	RS0	OV	-	P	0000 00x0B
ACC	Accumulator	E0H									0000 0000B
B	B Register	F0H									0000 0000B

Accumulator

ACC is the Accumulator register. The mnemonics for accumulator-specific instructions, however, refer to the accumulator simply as A.

B-Register

The B register is used during multiply and divide operations. For other instructions it can be treated as another scratch pad register.

Stack Pointer

The Stack Pointer register is 8 bits wide. It is incrementde before data is stored during PUSH and CALL executions. While the stack may reside anywhee in on-chip RAM, the Stack Pointer is initialized to 07H after a reset. This causes the stack to begin at location 08H.

Data Pointer

The Data Pointer (DPTR) consists of a high byte (DPH) and a low byte (DPL). Its intended function is to hold a 16-bit address. It may be manipulated as a 16-bit register or as two independent 8-bit registers.

Program Status Word(PSW)

The program status word(PSW) contains several status bits that reflect the current state of the CPU. The PSW, shown below, resides in the SFR space. It contains the Carry bit, the Auxiliary Carry(for BCD operation), the two register bank select bits, the Overflow flag, a Parity bit and two user-definable status flags.

The Carry bit, other than serving the function of a Carry bit in arithmetic operations, also serves as the "Accumulator" for a number of Boolean operations.

The bits RS0 and RS1 are used to select one of the four register banks shown in the previous page. A number of instructions refer to these RAM locations as R0 through R7.

The Parity bit reflects the number of 1s in the Accumulator. P=1 if the Accumulator contains an odd number of 1s and otherwise P=0.

PSW register

bit	B7	B6	B5	B4	B3	B2	B1	B0
name	CY	AC	F0	RS1	RS0	OV	-	P

CY : Carry flag.
AC : Auxilliary Carry Flag.(For BCD operations)
F0 : Flag 0.(Available to the user for general purposes)
RS1: Register bank select control bit 1.
RS0: Register bank select control bit 0.
OV : Overflow flag.
B1 : Reserved.
P : Parity flag.

5.2 Notes on Compatibility to Standard 80C51 MCU

SFR Name	SFR Address	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0X12	T1X12	-	-	-	-	-	-

T0X12

0 : The clock source of Timer 0 is Fosc/12.

1 : The clock source of Timer 0 is Fosc.

T1X12

0 : The clock source of Timer 1 is Fosc/12.

1 : The clock source of Timer 1 is Fosc.

SFR Name	SFR Address	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO	8FH	-	EX4	EX3	EX2	-	-	T1CLKO	T0CLKO

EX4

0 := Disable INT4 interrupt function.

1 := Enable INT4 interrupt function.

EX3

0 := Disable INT3 interrupt function.

1 := Enable INT3 interrupt function.

EX2

0 := Disable INT2 interrupt function.

1 := Enable INT2 interrupt function.

T1CLKO

0 := Disable TIMER1 overflow toggle P3.4.

1 := Enable TIMER1 overflow toggle P3.4.

T0CLKO

0 := Disable TIMER0 overflow toggle P3.5.

1 := Enable TIMER0 overflow toggle P3.5.

5.3 Addressing Modes

Addressing modes are an integral part of each computer's instruction set. They allow specifyng the source or destination of data in different ways, depending on the programming situation. There eight modes available:

- Register
- Direct
- Indirect
- Immediate
- Relative
- Absolute
- Long
- Indexed

Direct Addressing(DIR)
In direct addressing the operand is specified by an 8-bit address field in the instruction. Only internal data RAM and SFRs can be direct addressed.

Indirect Addressing(IND)
In indirect addressing the instruction specified a register which contains the address of the operand. Both internal and external RAM can be indirectly addressed.

The address register for 8-bit addresses can be R0 or R1 of the selected bank, or the Stack Pointer.
The address register for 16-bit addresses can only be the 16-bit data pointer register – DPTR.

Register Instruction(REG)
The register banks, containing registers R0 through R7, can be accessed by certain instructions which carry a 3-bit register specification within the opcode of the instruction. Instructions that access the registers this way are code efficient because this mode eliminates the need of an extra address byte. When such instruction is executed, one of the eight registers in the selected bank is accessed.

Register-Specific Instruction
Some instructions are specific to a certain register. For example, some instructions always operate on the accumulator or data pointer,etc. No address byte is needed for such instructions. The opcode itself does it.

Immediate Constant(IMM)
The value of a constant can follow the opcode in the program memory.

Index Addressing
Only program memory can be accessed with indexed addressing and it can only be read. This addressing mode is intended for reading look-up tables in program memory. A 16-bit base register(either DPTR or PC) points to the base of the table, and the accumulator is set up with the table entry number. Another type of indexed addressing is used in the conditional jump instruction.
In conditional jump, the destination address is computed as the sum of the base pointer and the accumulator.

5.4 Instruction Set Summary

The STC MCU instructions are fully compatible with the standard 8051's,which are divided among five functional groups:

- Arithmetic
- Logical
- Data transfer
- Boolean variable
- Program branching

The following tables provides a quick reference chart showing all the 8051 instructions. Once you are familiar with the instruction set, this chart should prove a handy and quick source of reference.

Mnemonic		Description	Byte	Execution clocks of conventional 8051	Execution clocks of STC15-28-PIN
ARITHMETIC OPERATIONS					
ADD	A, Rn	Add register to Accumulator	1	12	2
ADD	A, direct	Add ditect byte to Accumulator	2	12	3
ADD	A, @Ri	Add indirect RAM to Accumulator	1	12	3
ADD	A, #data	Add immediate data to Accumulator	2	12	2
ADDC	A, Rn	Add register to Accumulator with Carry	1	12	2
ADDC	A, direct	Add direct byte to Accumulator with Carry	2	12	3
ADDC	A, @Ri	Add indirect RAM to Accumulator with Carry	1	12	3
ADDC	A, #data	Add immediate data to Acc with Carry	2	12	2
SUBB	A, Rn	Subtract Register from Acc wih borrow	1	12	2
SUBB	A, direct	Subtract direct byte from Acc with borrow	2	12	3
SUBB	A, @Ri	Subtract indirect RAM from ACC with borrow	1	12	3
SUBB	A, #data	Substract immediate data from ACC with borrow	2	12	2
INC	A	Increment Accumulator	1	12	2
INC	Rn	Increment register	1	12	3
INC	direct	Increment direct byte	2	12	4
INC	@Ri	Increment direct RAM	1	12	4
DEC	A	Decrement Accumulator	1	12	2
DEC	Rn	Decrement Register	1	12	3
DEC	direct	Decrement direct byte	2	12	4
DEC	@Ri	Decrement indirect RAM	1	12	4
INC	DPTR	Increment Data Pointer	1	24	1
MUL	AB	Multiply A & B	1	48	4
DIV	AB	Divde A by B	1	48	5
DA	A	Decimal Adjust Accumulator	1	12	4

Mnemonic		Description	Byte	Execution clocks of conventional 8051	Execution clocks of STC15-28-PIN
LOGICAL OPERATIONS					
ANL	A, Rn	AND Register to Accumulator	1	12	2
ANL	A, direct	AND direct btye to Accumulator	2	12	3
ANL	A, @Ri	AND indirect RAM to Accumulator	1	12	3
ANL	A, #data	AND immediate data to Accumulator	2	12	2
ANL	direct, A	AND Accumulator to direct byte	2	12	4
ANL	direct,#data	AND immediate data to direct byte	3	24	4
ORL	A, Rn	OR register to Accumulator	1	12	2
ORL	A,direct	OR direct byte to Accumulator	2	12	3
ORL	A,@Ri	OR indirect RAM to Accumulator	1	12	3
ORL	A, #data	OR immediate data to Accumulator	2	12	2
ORL	direct, A	OR Accumulator to direct byte	2	12	4
ORL	direct,#data	OR immediate data to direct byte	3	24	4
XRL	A, Rn	Exclusive-OR register to Accumulator	1	12	2
XRL	A, direct	Exclusive-OR direct byte to Accumulator	2	12	3
XRL	A, @Ri	Exclusive-OR indirect RAM to Accumulator	1	12	3
XRL	A, #data	Exclusive-OR immediate data to Accumulator	2	12	2
XRL	direct, A	Exclusive-OR Accumulator to direct byte	2	12	4
XRL	direct,#data	Exclusive-OR immediate data to direct byte	3	24	4
CLR	A	Clear Accumulator	1	12	1
CPL	A	Complement Accumulator	1	12	2
RL	A	Rotate Accumulator Left	1	12	1
RLC	A	Rotate Accumulator Left through the Carry	1	12	1
RR	A	Rotate Accumulator Right	1	12	1
RRC	A	Rotate Accumulator Right through the Carry	1	12	1
SWAP	A	Swap nibbles within the Accumulator	1	12	1

Mnemonic		Description	Byte	Execution clocks of conventional 8051	Execution clocks of STC15-28-PIN
DATA TRANSFER					
MOV	A, Rn	Move register to Accumulator	1	12	1
MOV	A, direct	Move direct byte to Accumulator	2	12	2
MOV	A,@Ri	Move indirect RAM to	1	12	2
MOV	A, #data	Move immediate data to Accumulator	2	12	2
MOV	Rn, A	Move Accumulator to register	1	12	2
MOV	Rn, direct	Move direct byte to register	2	24	4
MOV	Rn, #data	Move immediate data to register	2	12	2
MOV	direct, A	Move Accumulator to direct byte	2	12	3
MOV	direct, Rn	Move register to direct byte	2	24	3
MOV	direct,direct	Move direct byte to direct	3	24	4
MOV	direct, @Ri	Move indirect RAM to direct byte	2	24	4
MOV	direct,#data	Move immediate data to direct byte	3	24	3
MOV	@Ri, A	Move Accumulator to indirect RAM	1	12	3
MOV	@Ri, direct	Move direct byte to indirect RAM	2	24	4
MOV	@Ri, #data	Move immediate data to indirect RAM	2	12	3
MOV	DPTR,#data16	Move immdiate data to indirect RAM	2	12	3
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to Acc	1	24	4
MOVC	A, @A+PC	Move Code byte relative to PC to Acc	1	24	4
MOVX	A,@Ri	Move External RAM(16-bit addr) to Acc	1	24	4
MOVX	A,@DPTR	Move External RAM(16-bit addr) to Acc	1	24	3
MOVX	@Ri, A	Move Acc to External RAM(8-bit addr)	1	24	3
MOVX	@DPTR,A	Move Acc to External RAM (16-bit addr)	1	24	3
PUSH	direct	Push direct byte onto stack	2	24	4
POP	direct	POP direct byte from stack	2	24	3
XCH	A,Rn	Exchange register with Accumulator	1	12	3
XCH	A, direct	Exchange direct byte with Accumulator	2	12	4
XCH	A, @Ri	Exchange indirect RAM with Accumulator	1	12	4
XCHD	A, @Ri	Exchange low-order Digit indirect RAM with Acc	1	12	4

Mnemonic		Description	Byte	Execution clocks of conventional 8051	Execution clocks of STC15-28-PIN
BOOLEAN VARIABLE MANIPULATION					
CLR	C	Clear Carry	1	12	1
CLR	bit	Clear direct bit	2	12	4
SETB	C	Set Carry	1	12	1
SETB	bit	Set direct bit	2	12	4
CPL	C	Complement Carry	1	12	1
CPL	bit	Complement direct bit	2	12	4
ANL	C, bit	AND direct bit to Carry	2	24	3
ANL	C, /bit	AND complement of direct bit to Carry	2	24	3
ORL	C, bit	OR direct bit to Carry	2	24	3
ORL	C, /bit	OR complement of direct bit to Carry	2	24	3
MOV	C, bit	Move direct bit to Carry	2	12	3
MOV	bit, C	Move Carry to direct bit	2	24	4
JC	rel	Jump if Carry is set	2	24	3
JNC	rel	Jump if Carry not set	2	24	3
JB	bit, rel	Jump if direct bit is set	3	24	4
JNB	bit,rel	Jump if direct bit is not set	3	24	4
JBC	bit, rel	Jump if direct bit is set & clear bit	3	24	5
PROGRAM BRANCHING					
ACALL	addr11	Absolute Subroutine Call	2	24	6
LCALL	addr16	Long Subroutine Call	3	24	6
RET		Return from Subroutine	1	24	4
RETI		Return from interrupt	1	24	4
AJMP	addr11	Absolute Jump	2	24	3
LJMP	addr16	Long Jump	3	24	4
SJMP	rel	Short Jump (relative addr)	2	24	3
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	24	3
JZ	rel	Jump if Accumulator is Zero	2	24	3
JNZ	rel	Jump if Accumulator is not Zero	2	24	3
CJNE	A,direct,rel	Compare direct byte to Acc and jump if not equal	3	24	5
CJNE	A,#data,rel	Compare immediate to Acc and Jump if not equal	3	24	4
CJNE	Rn,#data,rel	Compare immediate to register and Jump if not equal	3	24	4
CJNE	@Ri,#data,rel	Compare immediate to indirect and jump if not equal	3	24	5
DJNZ	Rn, rel	Decrement register and jump if not Zero	2	24	4
DJNZ	direct, rel	Decrement direct byte and Jump if not Zero	3	24	5
NOP		No Operation	1	12	1

Instruction execution speed boost summary:

24 times faster execution speed	1
12 times faster execution speed	12
9.6 times faster execution speed	1
8 times faster execution speed	20
6 times faster execution speed	39
4.8 times faster execution speed	4
4 times faster execution speed	20
3 times faster execution speed	14
24 times faster execution speed	1

Based on the analysis of frequency of use order statistics, STC 1T series MCU instruction execution speed is faster than the traditional 8051 MCU 8 ~ 12 times in the same working environment.

Instruction execution clock count:

1 clock instruction	12
2 clock instruction	20
3 clock instruction	38
4 clock instruction	34
5 clock instruction	5
6 clock instruction	2

5.5 Instruction Definitions

ACALL addr 11

Function: Absolute Call

Description: ACALL unconditionally calls a subroutine located at the indicated address.The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the Stack Pointer twice. The destination address is obtained by suceesively concatenating the five high-order bits of the incremented PC opcode bits 7-5,and the second byte of the instruction. The subroutine called must therefore start within the same 2K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

Example: Initially SP equals 07H. The label "SUBRTN" is at program memory location 0345H. After executingthe instruction,

ACALL SUBRTN

at location 0123H, SP will contain 09H, internal RAM locations 08H and 09H will contain 25H and 01H, respectively, and the PC will contain 0345H.

Bytes: 2

Cycles: 2

Encoding:

a10 a9 a8 1	0 0 1 0	a7 a6 a5 a4	a3 a2 a1 a0

Operation: ACALL
(PC)← (PC)+ 2
(SP)←(SP) + 1
((sP)) ← (PC_{7-0})
(SP)←(SP) + 1
((SP))←(PC_{15-8})
(PC_{10-0})← page address

ADD A,<src-byte>

Function: Add

Description: ADD adds the byte variable indicated to the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occured.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register,direct register-indirect, or immediate.

Example: The Accumulator holds 0C3H(11000011B) and register 0 holds 0AAH (10101010B). The instruction,

ADD A,R0

will leave 6DH (01101101B) in the Accumulator with the AC flag cleared and both the carry flag and OV set to 1.

ADD A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 1 0	1 r r r

Operation: ADD
(A)←(A) + (Rn)

ADD A,direct

Bytes: 2

Cycles: 1

Encoding:

0 0 1 0	0 1 0 1	direct address

Operation: ADD
(A)←(A) + (direct)

ADD A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 1 0	0 1 1 i

Operation: ADD
(A)←(A) + ((Ri))

ADD A,#data

Bytes: 2

Cycles: 1

Encoding:

0 0 1 0	0 1 0 0	immediate data

Operation: ADD
(A)←(A) + #data

ADDC A,<src-byte>

Function: Add with Carry

Description: ADDC simultaneously adds the byte variable indicated, the Carry flag and the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occured.
OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.
Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C3H(11000011B) and register 0 holds 0AAH (10101010B) with the Carry. The instruction,
ADDC A,R0
will leave 6EH (01101101B) in the Accumulator with the AC flag cleared and both the carry flag and OV set to 1.

ADDC A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 1 1	1 r r r

Operation: ADDC

(A)←(A) + (C) + (Rn)

ADDC A,direct

Bytes: 2

Cycles: 1

Encoding:

0 0 1 1	0 1 0 1	direct address

Operation: ADDC

(A)←(A) + (C) + (direct)

ADDC A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 1 1	0 1 1 i

Operation: ADDC

(A)←(A) + (C) + ((Ri))

ADDC A,#data

Bytes: 2

Cycles: 1

Encoding:

0 0 1 1	0 1 0 0	immediate data

Operation: ADDC

(A)←(A) + (C) + #data

AJMP addr 11

Function: Absolute Jump

Description: AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (after incrementing the PC twice), opcode bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block of program memory as the first byte of the instruction following AJMP.

Example: The label "JMPADR" is at program memory location 0123H. The instruction,

AJMP JMPADR

is at location 0345H and will load the PC with 0123H.

Bytes: 2

Cycles: 2

Encoding:

a10 a9 a8 0	0 0 0 1	a7 a6 a5 a4	a3 a2 a1 a0

Operation: AJMP

(PC)← (PC)+ 2

($PC_{10\text{-}0}$)← page address

ANL <dest-byte> , <src-byte>

Function: Logical-AND for byte variables

Description: ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch not the input pins.

Example: If the Accumulator holds 0C3H(11000011B) and register 0 holds 55H (01010101B) then the instruction,

ANL A,R0

will leave 41H (01000001B) in the Accumulator.

When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the Accumulator at run-time. The instruction,

ANL Pl, #01110011B

will clear bits 7, 3, and 2 of output port 1.

ANL A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 1 0 1	1 r r r

Operation: ANL
(A)←**(A)** ∧ (Rn)

ANL A,direct

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 1 0 1	direct address

Operation: ANL
(A)←**(A)** ∧ (direct)

ANL A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 1 0 1	0 1 1 i

Operation: ANL
(A)←**(A)** ∧ ((Ri))

ANL A,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 1 0 0	immediate data

Operation: ANL
(A)←**(A)** ∧ #data

ANL direct,A

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 0 1 0	direct address

Operation: ANL
(direct)←**(direct)** ∧ (A)

ANL direct,#data

Bytes: 3

Cycles: 2

Encoding:

0 1 0 1	0 0 1 1	direct address	immediate data

Operation: ANL
(direct)←**(direct)** ∧ #data

ANL C , <src-bit>

Function: Logical-AND for bit variables

Description: If the Boolean value of the source bit is a logical 0 then clear the carry flag; otherwise leave the carry flag in its current state. A slash (" / ") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, *but the source bit itself is not affceted.* No other flsgs are affected.

Only direct addressing is allowed for the source operand.

Example: Set the carry flag if, and only if, P1.0 = 1, ACC. 7 = 1, and OV = 0:

```
MOV   C, P1.0          ;LOAD CARRY WITH INPUT PIN STATE
ANL   C, ACC.7         ;AND CARRY WITH ACCUM. BIT.7
ANL   C, /OV           ;AND WITH INVERSE OF OVERFLOW FLAG
```

ANL C,bit

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 0 1 0	bit address

Operation: ANL
(C) ← **(C)** ∧ (bit)

ANL C, /bit

Bytes: 2

Cycles: 2

Encoding:

1 0 1 1	0 0 0 0	bit address

Operation: ADD

(C)←(C) ∧ $\overline{(bit)}$

CJNE <dest-byte>, <src-byte>, rel

Function: Compare and Jump if Not Equal

Description: CJNE compares the magnitudes of the first two operands, and branches if their values are not equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.

The first two operands allow four addressing mode combinations: the Accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

Example: The Accumulator contains 34H. Register 7 contains 56H. The first instruction in the sequence

```
                CJNE    R7,#60H, NOT-EQ
;               . . .   . . . . .       ; R7 = 60H.
NOT_EQ:         JC      REQ_LOW         ; IF R7 < 60H.
;               . . .   . . . . .       ; R7 > 60H.
```

sets the carry flag and branches to the instruction at label NOT-EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to Port 1 is also 34H, then the instruction,

WAIT: CJNE A,P1,WAIT

clears the carry flag and continues with the next instruction in sequence, since the Accumulator does equal the data read from P1. (If some other value was being input on Pl, the program will loop at this point until the P1 data changes to 34H.)

CJNE A,direct,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 0 1	direct address	rel. address

Operation:

(PC) ← (PC) + 3
IF (A) < > *(direct)*
THEN
    (PC) ← (PC) + *relative offset*
IF (A) < *(direct)*
THEN
    (C) ← 1
ELSE
    (C) ← 0

CJNE A,#data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 0 1	immediata data	rel. address

Operation:

(PC) ← **(PC) + 3**
IF (A) < > *(data)*
THEN
(PC) ← **(PC) +** *relative offset*
IF (A) < *(data)*
THEN
(C) ← **1**
ELSE
(C) ← **0**

CJNE Rn,#data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	1 r r r	immediata data	rel. address

Operation:

(PC) ← **(PC) + 3**
IF (Rn) < > *(data)*
THEN
(PC) ← **(PC) +** *relative offset*
IF (Rn) < *(data)*
THEN
(C) ← **1**
ELSE
(C) ← **0**

CJNE @Ri,#data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 1 i	immediate data	rel. address

Operation:

(PC) ← **(PC) + 3**
IF ((Ri)) < > *(data)*
THEN
(PC) ← **(PC) +** *relative offset*
IF ((Ri)) < *(data)*
THEN
(C) ← **1**
ELSE
(C) ← **0**

CLR A

Function: Clear Accumulator

Description: The Aecunmlator is cleared (all bits set on zero). No flags are affected.

Example: The Accumulator contains 5CH (01011100B). The instruction,

CLR A

will leave the Accumulator set to 00H (00000000B).

Bytes: 1

Cycles: 1

Encoding:

1 1 1 0	0 1 0 0

Operation: CLR
(A)← 0

CLR bit

Function: Clear bit

Description: The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.

Example: Port 1 has previously been written with 5DH (01011101B). The instruction,

CLR P1.2

will leave the port set to 59H (01011001B).

CLR C

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	0 0 1 1

Operation: CLR
(C) ← 0

CLR bit

Bytes: 2

Cycles: 1

Encoding:

1 1 0 0	0 0 1 0	bit address

Operation: CLR
(bit) ← 0

CPL A

Function: Complement Accumulator

Description: Each bit of the Accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to a zero and vice-versa. No flags are affected.

Example: The Accumulator contains 5CH(01011100B). The instruction,

```
CPL   A
```

will leave the Accumulator set to 0A3H (101000011B).

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1	0 1 0 0

Operation: CPL

$(A) \leftarrow \overline{(A)}$

CPL bit

Function: Complement bit

Description: The bit variable specified is complemented. A bit which had been a one is changed to zero and vice-versa. No other flags are affected. CLR can operate on the carry or any directly addressable bit.

Note:When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, not the input pin.

Example: Port 1 has previously been written with 5DH (01011101B). The instruction,

```
CLR   P1.1
CLR   P1.2
```

will leave the port set to 59H (01011001B).

CPL C

Bytes: 1

Cycles: 1

Encoding:

1 0 1 1	0 0 1 1

Operation: CPL

$(C) \leftarrow \overline{(C)}$

CPL bit

Bytes: 2

Cycles: 1

Encoding:

1 0 1 1	0 0 1 0	bit address

Operation: CPL

$(bit) \leftarrow \overline{(bit)}$

DA A

Function: Decimal-adjust Accumulator for Addition

Description: DA A adjusts the eight-bit value in the Accumulator resulting from the earlier addition of two variables (each in packed-BCD format), producing two four-bit digits.Any ADD or ADDC instruction may have been used to perform the addition.

If Accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the Accumulator producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

If the carry flag is now set or if the four high-order bits now exceed nine(1010xxxx-111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the Accumulator, depending on initial Accumulator and PSW conditions.

Note: DA A cannot simply convert a hexadecimal number in the Accumulator to BCD notation, nor does DA A apply to decimal subtraction.

Example: The Accumulator holds the value 56H(01010110B) representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67H (01100111B) representing the packed BCD digits of the decimal number 67.The carry flag is set. The instruction sequence.

```
ADDC   A,R3
DA     A
```

will first perform a standard twos-complement binary addition, resulting in the value 0BEH (10111110) in the Accumulator. The carry and auxiliary carry flags will be cleared.

The Decimal Adjust instruction will then alter the Accumulator to the value 24H (00100100B), indicating the packed BCD digits of the decimal number 24, the low-order two digits of the decimal sum of 56,67, and the carry-in. The carry flag will be set by the Decimal Adjust instruction, indicating that a decimal overflow occurred. The true sum 56, 67, and 1 is 124.

BCD variables can be incremented or decremented by adding 01H or 99H. If the Accumulator initially holds 30H (representing the digits of 30 decimal), then the instruction sequence,

```
ADD    A,#99H
DA     A
```

will leave the carry set and 29H in the Accumulator, since 30+99=129. The low-order byte of the sum can be interpreted to mean 30 – 1 = 29.

Bytes: 1

Cycles: 1

Encoding:

1 1 0 1	0 1 0 0

Operation: DA

-contents of Accumulator are BCD

IF [[(A_{3-0}) > 9] V [(AC) = 1]]

THEN(A_{3-0}) ← (**A_{3-0}**) + 6

AND

IF [[(A_{7-4}) > 9] V [(C) = 1]]

THEN (A_{7-4}) ← (**A_{7-4}**) + 6

DEC byte

Function: Decrement

Description: The variable indicated is decremented by 1. An original value of 00H will underflow to 0FFH.

No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: Register 0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H, respectively. The instruction sequence,

```
DEC   @R0
DEC   R0
DEC   @R0
```

will leave register 0 set to 7EH and internal RAM locations 7EH and 7FH set to 0FFH and 3FH.

DEC A

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	0 1 0 0

Operation: DEC

(A)←(**A**) −1

DEC Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	1 r r r

Operation: DEC

(Rn)←**(Rn) - 1**

DEC direct

Bytes: 2

Cycles: 1

Encoding:

0 0 0 1	0 1 0 1	direct address

Operation: DEC

(direct)←(direct) −1

DEC @Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	0 1 1 i

Operation: DEC

((Ri))←((Ri)) - 1

DIV AB

Function: Divide

Description: DIV AB divides the unsigned eight-bit integer in the Accumulator by the unsigned eight-bit integer in register B. The Accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared.

Exception: if B had originally contained 00H, the values returned in the Accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

Example: The Accumulator contains 251(OFBH or 11111011B) and B contains 18(12H or 00010010B). The instruction,

DIV AB

will leave 13 in the Accumulator (0DH or 00001101B) and the value 17 (11H or 00010010B) in B, since 251 = (13×18) + 17. Carry and OV will both be cleared.

Bytes: 1

Cycles: 4

Encoding:

1 0 0 0	0 1 0 0

Operation: DIV

$(A)_{15-8}$
$(B)_{7-0}$ ← (A)/(B)

DJNZ <byte>, <rel-addr>

Function: Decrement and Jump if Not Zero

Description: DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are afected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: Internal RAM locations 40H, 50H, and 60H contain the values 01H, 70H, and 15H, respectively. The instruction sequence,

```
DJNZ    40H, LABEL_1
DJNZ    50H, LABEL_2
DJNZ    60H, LABEL_3
```

will cause a jump to the instruction at label LABEL_2 with the values 00H, 6FH, and 15H in the three RAM locations. The first jump was not taken because the result was zero.

This instruction provides a simple way of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction The instruction sequence,

```
            MOV     R2,#8
TOOOLE:     CPL     P1.7
            DJNZ    R2, TOOGLE
```

will toggle P1.7 eight times, causing four output pulses to appear at bit 7 of output Port 1. Each pulse will last three machine cycles; two for DJNZ and one to alter the pin.

DJNZ Rn,rel

Bytes: 2

Cycles: 2

Encoding:

1 1 0 1	1 r r r	rel. address

Operation: DJNZ

(PC) ← (PC) + 2
(Rn) ← (Rn) – 1
IF (Rn) > 0 or (Rn) < 0
THEN
(PC) ← (PC)+ rel

DJNZ direct, rel

Bytes: 3

Cycles: 2

Encoding:

1 1 0 1	0 1 0 1	direct address	rel. address

Operation: DJNZ
(PC) ← **(PC) + 2**
(direct) ← **(direct) – 1**
IF (direct) > 0 or (direct) < 0
THEN
(PC) ← **(PC) + rel**

INC <byte>

Function: Increment

Description: INC increments the indicated variable by 1. An original value of 0FFH will overflow to 00H.No flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: Register 0 contains 7EH (011111110B). Internal RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. The instruction sequence,

```
INC   @R0
INC    R0
INC   @R0
```

will leave register 0 set to 7FH and internal RAM locations 7EH and 7FH holding (respectively) 00H and 41H.

INC A

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 1 0 0

Operation: INC
(A) ← **(A)+1**

INC Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	1 r r r

Operation: INC
(Rn) ← **(Rn)+1**

INC direct

Bytes: 2

Cycles: 1

Encoding:

0 0 0 0	0 1 0 1	direct address

Operation: INC
(direct)←**(direct)** + 1

INC @Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 1 1 i

Operation: INC
((Ri))←((Ri)) + 1

INC DPTR

Function: Increment Data Pointer

Description: Increment the 16-bit data pointer by 1. A 16-bit increment (modulo 2^{16}) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0FFH to 00H will increment the high-order-byte (DPH). No flags are affected.
This is the only 16-bit register which can be incremented.

Example: Register DPH and DPL contains 12H and 0FEH,respectively. The instruction sequence,
INC DPTR
INC DPTR
INC DPTR
will change DPH and DPL to 13H and 01H.

Bytes: 1

Cycles: 2

Encoding:

1 0 1 0	0 0 1 1

Operation: INC
(DPTR) ← (DPTR)+1

JB bit, rel

Function: Jump if Bit set

Description: If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified. No flags are affected.*

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56 (01010110B). The instruction sequence,
JB P1.2, LABEL1
JB ACC.2, LABEL2
will cause program execution to branch to the instruction at label LABEL2.

Bytes: 3

Cycles: 2

Encoding:

0 0 1 0	0 0 0 0	bit address	rel. address

Operation: JB
(PC) ← (PC)+ 3
IF (bit) = 1
THEN
(PC) ← (PC) + rel

JBC bit, rel

Function: Jump if Bit is set and Clear bit

Description: If the indicated bit is one,branch to the address indicated;otherwise proceed with the next instruction.*The bit wili not be cleared if it is already a zero.* The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

Note: When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, not the input pin.

Example: The Accumulator holds 56H (01010110B). The instruction sequence,

```
JBC     ACC.3, LABEL1
JBC     ACC.2, LABEL2
```

will cause program execution to continue at the instruction identified by the label LABEL2, with the Accumulator modified to 52H (01010010B).

Bytes: 3

Cycles: 2

Encoding:

0 0 0 1	0 0 0 0	bit address	rel. address

Operation:

```
JBC
(PC) ← (PC)+ 3
IF  (bit) = 1
    THEN
          (bit) ← 0
          (PC) ← (PC) + rel
```

JC rel

Function: Jump if Carry is set

Description: If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice.No flags are affected.

Example: The carry flag is cleared. The instruction sequence,

```
JC      LABEL1
CPL     C
JC      LABEL2s
```

will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 0 0	0 0 0 0	rel. address

Operation:

```
JC
(PC) ← (PC)+ 2
IF  (C) = 1
    THEN
          (PC) ← (PC) + rel
```

JMP @A+DPTR

Function: Jump indirect

Description: Add the eight-bit unsigned contents of the Accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 2^{16}): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the Accumulator nor the Data Pointer is altered. No flags are affected.

Example: An even number from 0 to 6 is in the Accumulator. The following sequence of instructions will branch to one of four AJMP instructions in a jump table starting at JMP_TBL:

```
            MOV     DPTR, #JMP_TBL
            JMP     @A+DPTR
JMP-TBL:    AJMP    LABEL0
            AJMP    LABEL1
            AJMP    LABEL2
            AJMP    LABEL3
```

If the Accumulator equals 04H when starting this sequence, execution will jump to label LABEL2. Remember that AJMP is a two-byte instruction, so the jump instructions start at every other address.

Bytes: 1

Cycles: 2

Encoding:

0 1 1 1	0 0 1 1

Operation: JMP

(PC) ← (A) + (DPTR)

JNB bit, rel

Function: Jump if Bit is not set

Description: If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56H (01010110B). The instruction sequence,

```
JNB     P1.3, LABEL1
JNB     ACC.3, LABEL2
```

will cause program execution to continue at the instruction at label LABEL2

Bytes: 3

Cycles: 2

Encoding:

0 0 1 1	0 0 0 0	bit address	rel. address

Operation: JNB

(PC) ← (PC)+ 3

IF (bit) = 0

THEN (PC) ← (PC) + rel

JNC rel

Function: Jump if Carry not set

Description: If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified

Example: The carry flag is set. The instruction sequence,

```
JNC   LABEL1
CPL   C
JNC   LABEL2
```

will clear the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 0 1	0 0 0 0	rel. address

Operation:

```
JNC
(PC) ← (PC)+ 2
IF  (C) = 0
    THEN    (PC) ← (PC) + rel
```

JNZ rel

Function: Jump if Accumulator Not Zero

Description: If any bit of the Accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

Example: The Accumulator originally holds 00H. The instruction sequence,

```
JNZ    LABEL1
INC    A
JNZ    LAEEL2
```

will set the Accumulator to 01H and continue at label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 1 1	0 0 0 0	rel. address

Operation:

```
JNZ
(PC) ← (PC)+ 2
IF  (A) ≠ 0
    THEN    (PC) ← (PC) + rel
```

JZ rel

Function: Jump if Accumulator Zero

Description: If all bits of the Accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

Example: The Accumulator originally contains 01H. The instruction sequence,
JZ LABEL1
DEC A
JZ LAEEL2
will change the Accumulator to 00H and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 1 0	0 0 0 0	rel. address

Operation: JZ
(PC) ← (PC)+ 2
IF (A) = 0
THEN (PC) ← (PC) + rel

LCALL addr16

Function: Long call

Description: LCALL calls a subroutine loated at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the Stack Pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64K-byte program memory address space. No flags are affected.

Example: Initially the Stack Pointer equals 07H. The label “SUBRTN” is assigned to program memory location 1234H. After executing the instruction,

LCALL SUBRTN

at location 0123H, the Stack Pointer will contain 09H, internal RAM locations 08H and 09H will contain 26H and 01H, and the PC will contain 1234H.

Bytes: 3

Cycles: 2

Encoding:

0 0 0 1	0 0 1 0	addr15-addr8	addr7-addr0

Operation: LCALL
(PC) ← (PC) + 3
(SP) ← (SP) + 1
((SP)) ← (PC_{7-0})
(SP) ← (SP) + 1
((SP)) ← (PC_{15-8})
(PC) ← $addr_{15-0}$

LJMP addr16

Function: Long Jump

Description: LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.

Example: The label "JMPADR" is assigned to the instruction at program memory location 1234H. The instruction,

LJMP JMPADR

at location 0123H will load the program counter with 1234H.

Bytes: 3

Cycles: 2

Encoding:

0 0 0 0	0 0 1 0	addr15-addr8	addr7-addr0

Operation: LJMP
(PC) ← $addr_{15-0}$

MOV <dest-byte> , <src-byte>

Function: Move byte variable

Description: The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.

This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.

Example: Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (0CAH).

```
MOV     R0, #30H    ;R0< = 30H
MOV     A, @R0      ;A < = 40H
MOV     R1, A       ;R1 < = 40H
MOV     B, @Rl      ;B < = 10H
MOV     @Rl, Pl     ;RAM (40H) < = 0CAH
MOV     P2, P1      ;P2  #0CAH
```

leaves the value 30H in register 0,40H in both the Accumulator and register 1,10H in register B, and 0CAH(11001010B) both in RAM location 40H and output on port 2.

MOV A,Rn

Bytes: 1

Cycles: 1

Encoding:

1 1 1 0	1 r r r

Operation: MOV
(A) ← **(Rn)**

***MOV A,direct**

Bytes: 2

Cycles: 1

Encoding:

1 1 1 0	0 1 0 1	direct address

Operation: MOV
(A)← (direct)

***MOV A, ACC is not a valid instruction**

MOV A,@Ri

Bytes: 1

Cycles: 1

Encoding:

1 1 1 0	0 1 1 i

Operation: MOV
(A) ← ((Ri))

MOV A,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 1	0 1 0 0	immediate data

Operation: MOV
(A)← #data

MOV Rn, A

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1	1 r r r

Operation: MOV
(Rn)←(A)

MOV Rn,direct

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0	1 r r r	direct addr.

Operation: MOV
(Rn)←(direct)

MOV Rn,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 1	1 r r r	immediate data

Operation: MOV
(Rn) ← #data

MOV direct, A

Bytes: 2

Cycles: 1

Encoding:

1 1 1 1	0 1 0 1	direct address

Operation: MOV
(direct) ← (A)

MOV direct, Rn

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	1 r r r	direct address

Operation: MOV
(direct) ← (Rn)

MOV direct, direct

Bytes: 3

Cycles: 2

Encoding:

1 0 0 0	0 1 0 1	dir.addr. (src)

Operation: MOV
(direct)← (direct)

MOV direct, @Ri

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 1 1 i	direct addr.

Operation: MOV
(direct)←((Ri))

MOV direct,#data

Bytes: 3

Cycles: 2

Encoding:

0 1 1 1	0 1 0 1	direct address

Operation: MOV
(direct) ← #data

MOV @Ri, A

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1	0 1 1 i

Operation: MOV
((Ri)) ← (A)

MOV @Ri, direct

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0	0 1 1 i	direct addr.

Operation: MOV
((Ri)) ← (direct)

MOV @Ri, #data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 1	0 1 1 i	immediate data

Operation: MOV
((Ri)) ← #data

MOV <dest-bit> , <src-bit>

Function: Move bit data

Description: The Boolean variable indicated by the second operand is copied into the location specified by the first operand. One of the operands must be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.

Example: The carry flag is originally set. The data present at input Port 3 is 11000101B. The data previously written to output Port 1 is 35H (00110101B).

```
MOV     P1.3, C
MOV     C, P3.3
MOV     P1.2, C
```

will leave the carry cleared and change Port 1 to 39H (00111001B).

MOV C,bit

Bytes: 2

Cycles: 1

Encoding:

1 0 1 0	0 0 1 1	bit address

Operation: MOV
(C) ← (bit)

MOV bit,C

Bytes: 2

Cycles: 2

Encoding:

1 0 0 1	0 0 1 0	bit address

Operation: MOV
(bit)← (C)

MOV DPTR , #data 16

Function: Load Data Pointer with a 16-bit constant

Description: The Data Pointer is loaded with the 16-bit constant indicated.The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected.
This is the only instruction which moves 16 bits of data at once.

Example: The instruction,
MOV DPTR, #1234H
will load the value 1234H into the Data Pointer: DPH will hold 12H and DPL will hold 34H.

Bytes: 3

Cycles: 2

Encoding:

1 0 0 1	0 0 0 0	immediate data 15-8

Operation: MOV
(DPTR) ← #data$_{15\text{-}0}$
DPH DPL ← #data$_{15\text{-}8}$ #data$_{7\text{-}0}$

MOVC A , @A+ <base-reg>

Function: Move Code byte

Description: The MOVC instructions load the Accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit. Accumulator contents and the contents of a sixteen-bit base register, which may be either the Data Pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the Accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

Example: A value between 0 and 3 is in the Accumulator. The following instructions will translate the value in the Accumulator to one of four values defimed by the DB (define byte) directive.

```
REL-PC:  INC     A
         MOVC    A, @A+PC
         RET
         DB      66H
         DB      77H
         DB      88H
         DB      99H
```

If the subroutine is called with the Accumulator equal to 01H, it will return with 77H in the Accumulator. The INC A before the MOVC instruction is needed to “get around” the RET instruction above the table. If several bytes of code separated the MOVC from the table, the corresponding number would be added to the Accumulator instead.

MOVC A,@A+DPTR

Bytes: 1

Cycles: 2

Encoding:

1 0 0 1	0 0 1 1

Operation: MOVC
(A) ← ((A)+(DPTR))

MOVC A,@A+PC

Bytes: 1

Cycles: 2

Encoding:

1 0 0 0	0 0 1 1

Operation: MOVC
(PC) ← (PC)+1
(A) ← ((A)+(PC))

MOVX <dest-byte> , <src-byte>

Function: Move External

Description: The MOVX instructions transfer data between the Accumulator and a byte of external data memory, hence the "X" appended to MOV. There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.

In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or for a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instruction, the Data Pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.

It is possible in some situations to mix the two MOVX types. A large RAM array with its high-order address lines driven by P2 can be addressed via the Data Pointer, or with code to output high-order address bits to P2 followed by a MOVX instruction using R0 or R1.

Example: An external 256 byte RAM using multiplexed address/data lines (e.g., an Intel 8155 RAM/I/O/Timer) is connected to the 8051 Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12H and 34H. Location 34H of the external RAM holds the value 56H. The instruction sequence,

```
MOVX    A, @R1
MOVX    @R0, A
```

copies the value 56H into both the Accumulator and external RAM location 12H.

MOVX A,@Ri

Bytes: 1

Cycles: 2

Encoding:

1 1 1 0	0 0 1 i

Operation: MOVX
(A) ← ((Ri))

MOVX A,@DPTR

Bytes: 1

Cycles: 2

Encoding:

1 1 1 0	0 0 0 0

Operation: MOVX
(A) ← ((DPTR))

MOVX @Ri, A

Bytes: 1

Cycles: 2

Encoding:

1 1 1 1	0 0 1 i

Operation: MOVX
((Ri))← (A)

MOVX @DPTR, A

Bytes: 1

Cycles: 2

Encoding:

1 1 1 1	0 0 0 0

Operation: MOVX
(DPTR)←(A)

MUL AB

Function: Multiply

Description: MUL AB multiplies the unsigned eight-bit integers in the Accumulator and register B. The low-order byte of the sixteen-bit product is left in the Accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared

Example: Originally the Accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H). The instruction,

MUL AB

will give the product 12,800 (3200H), so B is changed to 32H (00110010B) and the Accumulator is cleared. The overflow flag is set, carry is cleared.

Bytes: 1

Cycles: 4

Encoding:

1 0 1 0	0 1 0 0

Operation: MUL
$(A)_{7\text{-}0}$ ← (A)×(B)
$(B)_{15\text{-}8}$

NOP

Function: No Operation

Description: Execution continues at the following instruction. Other than the PC, no registers or flags are affected.

Example: It is desired to produce a low-going output pulse on bit 7 of Port 2 lasting exactly 5 cycles. A simple SETB/CLR sequence would generate a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the instruction sequence.

```
CLR     P2.7
NOP
NOP
NOP
NOP
SETB    P2.7
```

Bytes: 1

Cycles: 1

Encoding:

0	0	0	0	0	0	0	0

Operation: NOP
(PC) ← (PC)+1

ORL <dest-byte> , <src-byte>

Function: Logical-OR for byte variables

Description: ORL performs the bitwise logical-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: If the Accumulator holds 0C3H (11000011B) and R0 holds 55H (01010101B) then the instruction,

```
ORL     A, R0
```

will leave the Accumulator holding the value 0D7H (11010111B).
When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be either a constant data value in the instruction or a variable computed in the Accumulator at run-time.The instruction,

```
ORL     P1, #00110010B
```

will set bits 5,4, and 1of output Port 1.

ORL A,Rn

Bytes: 1

Cycles: 1

Encoding:

0	1	0	0	1	r	r	r

Operation: ORL
(A) ← **(A)**∨(Rn)

ORL A,direct

Bytes: 2

Cycles: 1

Encoding:

0	1	0	0	0	1	0	1	direct address

Operation: ORL
(A)← **(A)**∨(direct)

ORL A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0	1	0	0	0	1	1	i

Operation: ORL
(A)← **(A)**∨((Ri))

ORL A,#data

Bytes: 2

Cycles: 1

Encoding:

0	1	0	0	0	1	0	0	immediate data

Operation: ORL
(A)← **(A)**∨ #data

ORL direct, A

Bytes: 2

Cycles: 1

Encoding:

0	1	0	0	0	0	1	0	direct address

Operation: ORL
(direct)← **(direct)**∨(A)

ORL direct, #data

Bytes: 3

Cycles: 2

Encoding:

0	1	0	0	0	0	1	1	direct address	immediate data

Operation: ORL
(direct) ← **(direct)**∨#data

ORL C, <src-bit>

Function: Logical-OR for bit variables

Description: Set the carry flag if the Boolean value is a logical 1; leave the carry in its current state otherwise. A slash (" / ") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

Example: Set the carry flag if and only if P1.0 = 1, ACC. 7 = 1, or OV = 0:

```
MOV     C, P1.0         ;LOAD CARRY WITH INPUT PIN P10
ORL     C, ACC.7        ;OR CARRY WITH THE ACC.BIT 7
ORL     C, /OV          ;OR CARRY WITH THE INVERSE OF OV
```

ORL C, bit

Bytes: 2

Cycles: 2

Encoding:

0 1 1 1	0 0 1 0	bit address

Operation: ORL

(C) ← (C)∨(bit)

ORL C, /bit

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0	0 0 0 0	bit address

Operation: ORL

(C) ← (C)∨$\overline{(bit)}$

POP direct

Function: Pop from stack

Description: The contents of the internal RAM location addressed by the Stack Pointer is read, and the Stack Pointer is decremented by one. The value read is then transferred to the directly addressed byte indicated. No flags are affected.

Example: The Stack Pointer originally contains the value 32H, and internal RAM locations 30H through 32H contain the values 20H, 23H, and 01H, respectively. The instruction sequence,

```
POP     DPH
POP     DPL
```

will leave the Stack Pointer equal to the value 30H and the Data Pointer set to 0123H. At this point the instruction,

```
POP     SP
```

will leave the Stack Pointer set to 20H. Note that in this special case the Stack Pointer was decremented to 2FH before being loaded with the value popped (20H).

Bytes: 2

Cycles: 2

Encoding:

1 1 0 1	0 0 0 0	direct address

Operation: POP

(diect) ← ((SP))

(SP) ← (SP) - 1

PUSH direct

Function: Push onto stack

Description: The Stack Pointer is incremented by one. The contents of the indicated variableis then copied into the internal RAM location addressed by the Stack Pointer. Otherwise no flags are affected.

Example: On entering interrupt routine the Stack Pointer contains 09H. The Data Pointer holds the value 0123H. The instruction sequence,

```
PUSH    DPL
PUSH    DPH
```

will leave the Stack Pointer set to 0BH and store 23H and 01H in internal RAM locations 0AH and 0BH, respectively.

Bytes: 2

Cycles: 2

Encoding:

1 1 0 0	0 0 0 0	direct address

Operation: PUSH

(SP) ← (SP) + 1

((SP)) ← (direct)

RET

Function: Return from subroutine

Description: RET pops the high-and low-order bytes of the PC successively from the stack, decrementing the Stack Pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.

Example: The Stack Pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

```
RET
```

will leave the Stack Pointer equal to the value 09H. Program execution will continue at location 0123H.

Bytes: 1

Cycles: 2

Encoding:

0 0 1 0	0 0 1 0

Operation: RET

(PC_{15-8}) ← ((SP))

(SP) ← (SP) -1

(PC_{7-0}) ← ((SP))

(SP) ← (SP) -1

RETI

Function: Return from interrupt

Description: RETI pops the high- and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The Stack Pointer is left decremented by two. No other registers are affected; the PSW is not automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt had been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.

Example: The Stack Pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RETI

will leave the Stack Pointer equal to 09H and return program execution to location 0123H.

Bytes: 1

Cycles: 2

Encoding:

0	0	1	1	0	0	1	0

Operation: RETI
$(PC_{15-8}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$
$(PC_{7-0}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$

RL A

Function: Rotate Accumulator Left

Description: The eight bits in the Accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

RL A

leaves the Accumulator holding the value 8BH (10001011B) with the carry unaffected.

Bytes: 1

Cycles: 1

Encoding:

0	0	1	0	0	0	1	1

Operation: RL
$(An+1) \leftarrow (An)$ n = 0-6
$(A0) \leftarrow (A7)$

RLC A

Function: Rotate Accumulator Left through the Carry flag

Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,
RLC A
leaves the Accumulator holding the value 8BH (10001011B) with the carry set.

Bytes: 1

Cycles: 1

Encoding:

0	0	1	1	0	0	1	1

Operation: RLC
(An+1) ← (An) n = 0-6
(A0) ← (C)
(C) ← (A7)

RR A

Function: Rotate Accumulator Right

Description: The eight bits in the Accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,
RR A
leaves the Accumulator holding the value 0E2H (11100010B) with the carry unaffected.

Bytes: 1

Cycles: 1

Encoding:

0	0	0	0	0	0	1	1

Operation: RR
(An) ← (An+1) n = 0 - 6
(A7) ← (A0)

RRC A

Function: Rotate Accumulator Right through the Carry flag

Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position.No other flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,
RRC A
leaves the Accumulator holding the value 62H (01100010B) with the carry set.

Bytes: 1

Cycles: 1

Encoding:

0	0	0	1	0	0	1	1

Operation: RRC
(An+1) ← (An) n = 0-6
(A7) ← (C)
(C) ← (A0)

SETB <bit>

Function: Set bit

Description: SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected

Example: The carry flag is cleared. Output Port 1 has been written with the value 34H (00110100B). The instructions,

```
SETB    C
SETB    P1.0
```

will leave the carry flag set to 1 and change the data output on Port 1 to 35H (00110101B).

SETB C

Bytes: 1

Cycles: 1

Encoding:

1 1 0 1	0 0 1 1

Operation: SETB

(C) ← 1

SETB bit

Bytes: 2

Cycles: 1

Encoding:

1 1 0 1	0 0 1 0	bit address

Operation: SETB

(bit) ← 1

SJMP rel

Function: Short Jump

Description: Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128bytes preceding this instruction to 127 bytes following it.

Example: The label "RELADR" is assigned to an instruction at program memory location 0123H. The instruction,

```
SJMP    RELADR
```

will assemble into location 0100H. After the instruction is executed, the PC will contain the value 0123H.

(*Note:* Under the above conditions the instruction following SJMP will be at 102H.Therefore, the displacement byte of the instruction will be the relative offset (0123H - 0102H) = 21H. Put another way, an SJMP with a displacement of 0FEH would be an one-instruction infinite loop).

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 0 0 0	rel. address

Operation: SJMP

(PC) ← (PC)+2

(PC) ← (PC)+rel

SUBB A, <src-byte>

Function: Subtract with borrow

Description: SUBB subtracts the indicated variable and the carry flag together from the Accumulator, leaving the result in the Accumulator. SUBB sets the carry (borrow)flag if a borrow is needed for bit 7, and clears C otherwise.(If C was set before executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the Accumulator along with the source operand).AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C9H (11001001B), register 2 holds 54H (01010100B), and the carry flag is set. The instruction,

SUBB A, R2

will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Notice that 0C9H minus 54H is 75H. The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be explicitly cleared by a CLR C instruction.

SUBB A, Rn

Bytes: 1

Cycles: 1

Encoding:

1 0 0 1	1 r r r

Operation: SUBB

(A) ← (A) - (C) - (Rn)

SUBB A, direct

Bytes: 2

Cycles: 1

Encoding:

1 0 0 1	0 1 0 1	direct address

Operation: SUBB

(A) ← (A) - (C) - (direct)

SUBB A, @Ri

Bytes: 1

Cycles: 1

Encoding:

1 0 0 1	0 1 1 i

Operation: SUBB

(A) ← (A) - (C) - ((Ri))

SUBB A, #data

Bytes: 2

Cycles: 1

Encoding:

1 0 0 1	0 1 0 0	immediate data

Operation: SUBB

(A) ← (A) - (C) - #data

SWAP A

Function: Swap nibbles within the Accumulator

Description: SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the Accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

SWAP A

leaves the Accumulator holding the value 5CH (01011100B).

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	0 1 0 0

Operation: SWAP

$(A_{3-0}) \rightleftarrows (A_{7-4})$

XCH A, <byte>

Function: Exchange Accumulator with byte variable

Description: XCH loads the Accumulator with the contents of the indicated variable, at the same time writing the original Accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.

Example: R0 contains the address 20H. The Accumulator holds the value 3FH (00111111B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCH A, @R0

will leave RAM location 20H holding the values 3FH (00111111B) and 75H (01110101B) in the accumulator.

XCH A, Rn

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	1 r r r

Operation: XCH

(A) ⇄ (Rn)

XCH A, direct

Bytes: 2

Cycles: 1

Encoding:

1 1 0 0	0 1 0 1	direct address

Operation: XCH

(A) ⇄ (direct)

XCH A, @Ri

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	0	1	1	i

Operation: XCH

(A) $\rightleftarrows$ ((Ri))

XCHD A, @Ri

Function: Exchange Digit

Description: XCHD exchanges the low-order nibble of the Accumulator (bits 3-0), generally representing a hexadecimal or BCD digit, with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.

Example: R0 contains the address 20H. The Accumulator holds the value 36H (00110110B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCHD A, @R0

will leave RAM location 20H holding the value 76H (01110110B) and 35H (00110101B) in the accumulator.

Bytes: 1

Cycles: 1

Encoding:

1	1	0	1	0	1	1	i

Operation: XCHD

(A_{3-0}) $\rightleftarrows$ (Ri_{3-0})

XRL <dest-byte>, <src-byte>

Function: Logical Exclusive-OR for byte variables

Description: XRL performs the bitwise logical Exclusive-OR operation between the indicated variables, storing the results in the destination. No flags are affected.

The two operands allow six addressing mode combinations.When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address,the source can be the Accumulator or immediate data.

(*Note*: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.)

Example: If the Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

XRL A, R0

will leave the Accumulator holding the vatue 69H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combinnation of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the Accumulator at run-time. The instruction,

XRL P1, #00110001B

will complement bits 5,4 and 0 of outpue Port 1.

XRL A, Rn

Bytes: 1

Cycles: 1

Encoding:

0 1 1 0	1 r r r

Operation: XRL
(A) ← (A) ⅄ (Rn)

XRL A, direct

Bytes: 2

Cycles: 1

Encoding:

0 1 1 0	0 1 0 1	direct address

Operation: XRL
(A) ← (A) ⅄ (direct)

XRL A, @Ri

Bytes: 1

Cycles: 1

Encoding:

0 1 1 0	0 1 1 i

Operation: XRL
(A) ← (A) ⅄ ((Ri))

XRL A, #data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 0	0 1 0 0	immediate data

Operation: XRL
(A) ← (A) ⅄ #data

XRL direct, A

Bytes: 2

Cycles: 1

Encoding:

0 1 1 0	0 0 1 0	direct address

Operation: XRL
(direct) ← (direct) ⅄ (A)

XRL direct, #dataw

Bytes: 3

Cycles: 2

Encoding:

0 1 1 0	0 0 1 1	direct address	immediate data

Operation: XRL
(direct) ← (direct) ⅄ # data

Chapter 6 Interrupts

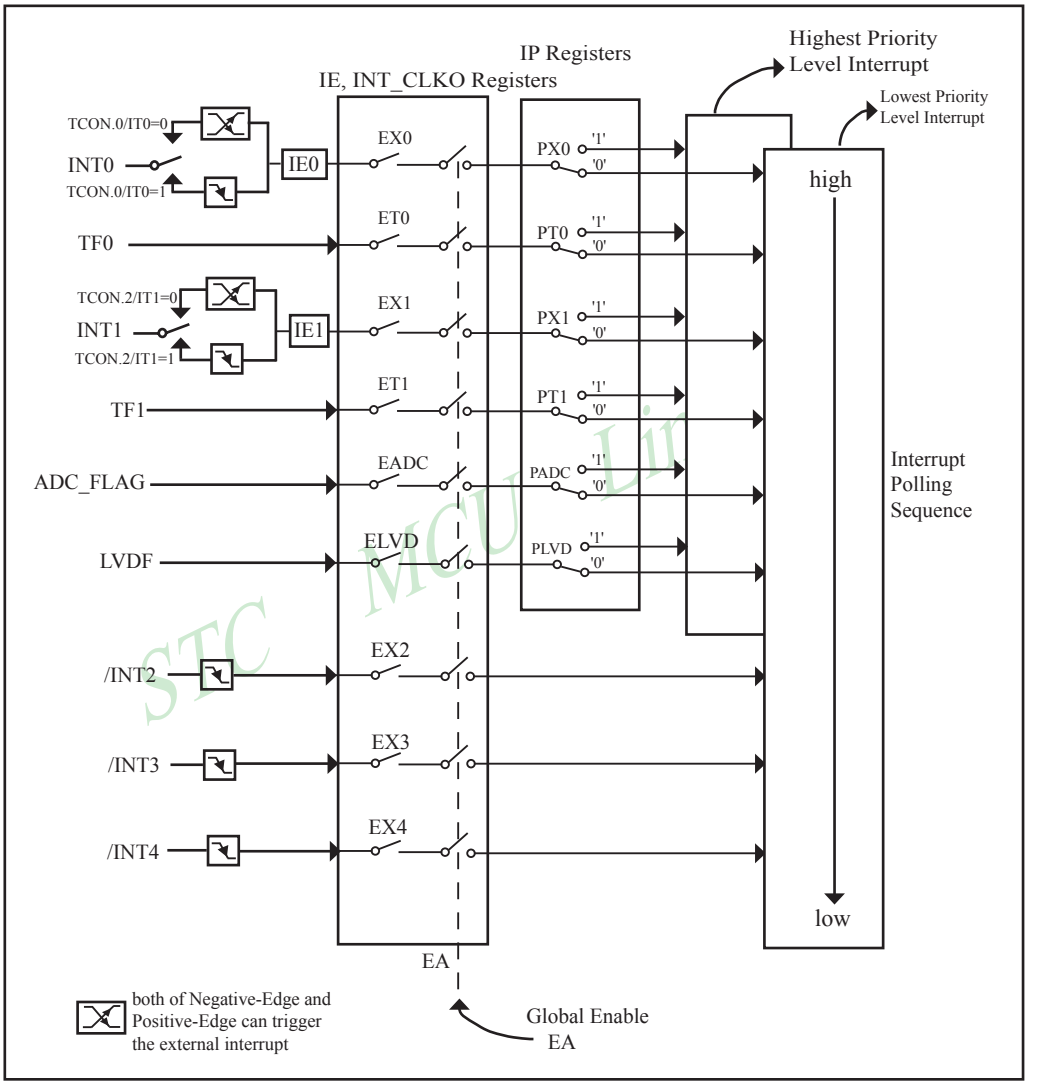
There are 9 interrupt vector addresses available in STC15-28-PIN. Associating with each interrupt vector, the interrupt sources can be individually enabled or disabled by setting or clearing a bit in the registers IE, INT_CLKO. These registers also contains a global disable bit(EA), which can be cleared to disable all interrupts at once.

All interrupt sources, except external interrupt 2 and external interrupt 3 and external interrupt 4, have one corresponding bit to represent its priority, which is located in SFR named IP register. Higher-priority interrupt will be not interrupted by lower-priority interrupt request. If two interrupt requests of different priority levels are received simultaneously, the request of higher priority is serviced. If interrupt requests of the same priority level are received simultaneously, an internal polling sequence determine which request is serviced. The following table shows the internal polling sequence in the same priority level and the interrupt vector address.

Interrupt Table

Interrupt Source	Vector address	Priority	Priority within level	Interrupt Request	Interrupt Enable Control Bit
External interrupt 0	0003H	0/1	1 st(highest)	IE0	EX0/EA
Timer 0	000BH	0/1	2nd	TF0	ET0/EA
External interrupt 1	0013H	0/1	3 rd	IE1	EX1/EA
Timer1	001BH	0/1	4 th	TF1	ET1/EA
	0023B				
ADC	002BH	0/1	5 th	ADC_FLAG	EADC/EA
LVD	0033H	0/1	6 th	LVDF	ELVD/EA
	003BH				
	0043H				
	004BH				
INT2	0053H	0	7 th		EX2/EA
INT3	005BH	0	8 th		EX3/EA
	0063H				
	006BH				
	0073H				
	007BH				
INT4	0083H	0	9 th		EX4/EA

6.1 Interrupt Structure



STC15-28-PIN Interrupt system diagram

The External Interrupts INT0 and INT1 can each be either negative-edge-activated or positive-edge-activated, depending on bits IT0 and IT1 in Register TCON. When ITx (x=0 or 1) is set, the external interrupts INTx (x=0 or 1) can be negative-edge-activated. When ITx (x=0 or 1) is cleared, both of Negative-Edge and Positive-Edge can trigger the external interrupt INTx(x=0 or 1). The flags that actually generate these interrupts are bits IE0 and IE1 in TCON.The interrupt flag will automatically cleared after interrupt acknowledge.
The interrupt from INTx (x=0,1) can trigger interrupt as well as wakes up CPU from power-down mode.

The Timer 0 and Timer1 Interrupts are generated by TF0 and TF1, which are set by a rollover in their respective Timer/Counter registers in most cases. When a timer interrupt is generated, the flag that generated it is cleared by the on-chip hardware when the service routine is vectored to.

The ADC interrupt is generated by the flag – ADC_FLAG (ADC_CONTR.4). It should be cleared by software.

The Low Voltage Detect interrupt is generated by the flag – LVDF(PCON.5) in PCON register. It should be cleared by software.

The External Interrupts $\overline{\text{INT2}}$ ~ $\overline{\text{INT4}}$ only can be negative-edge-activated. The interrupt flag is implied, not user acceptable. The interrupt flag will be cleared after interrupt acknowledge or EXn (n=2,3,4) goes low.
The interrupt from $\overline{\text{INTx}}$ (x=2,3,4) can trigger interrupt as well as wakes up CPU from power-down mode.

All of the bits that generate interrupts can be set or cleared by software, with the same result as though it had been set or cleared by hardware. In other words, interrupts can be generated or pending interrupts can be canceled in software.

Interrupt Trigger

Source	Trigger Moment
External interrupt 0	(IT0==1):= Negative-Edge (IT0==0):=Negative-Edge and Positive-Edge
Timer 0	TIMER0 overflow
External interrupt 1	(IT1==1):= Negative-Edge (IT1==0):=Negative-Edge and Positive-Edge
Timer1	TIMER1 overflow
LVD	Power drops under LVD-setting level
INT2	Negative-Edge
INT3	Negative-Edge
INT4	Negative-Edge

6.2 Interrupt Register

Symbol	Description	Address	Bit Address and Symbol MSB LSB								Value after Power-on or Reset
IE	Interrupt Enable	A8H	EA	ELVD	EADC	-	ET1	EX1	ET0	EX0	000x 0000B
IP	Interrupt Priority Low	B8H	-	PLVD	PADC	-	PT1	PX1	PT0	PX0	x00x 0000B
TCON	Timer Control register	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000 0000B
PCON	Power Control register	87H	-	-	LVDF	-	-	-	PD	IDL	xx1x xx00B
INT_CLKO	External Interrupt enable and Clock output register	8FH	-	EX4	EX3	EX2	-	-	T1CLKO	T0CLKO	x000 xx00B
ADC_CONTR	ADC control register	BCH	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHIS0	0000 0000B

IE: Interrupt Enable Rsgister

SFR Name	SFR Address	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	EA	ELVD	EADC	-	ET1	EX1	ET0	EX0

EA : disables all interrupts. if EA = 0,no interrupt will be acknowledged. if EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

ELVD : Low volatge detection interrupt enable
0 : = Disable Voltage Drop interrupt
1 : = Enable Voltage Drop interrupt.

EADC : ADC interrupt enable bit
0 : = Disable ADC interrupt
1 : = Enable ADC interrupt.

ET1 : Timer 1 interrupt enable bit
0 : = Disable TIMER1 interrupt
1 : = Enable TIMER1 interrupt.

EX1 : External interrupt 1 enable bit
0 : = Disable INT1 interrupt
1 : = Enable INT1 interrupt.
A Negative-Edge from INT1 pin will trigger an interrupt if IT1 (TCON.2) is set, and both of Negative-Edge and Positive-Edge will trigger an interrupt if IT1(TCON.2) is cleared. The interrupt flag IE1(TCON.3) will automatically cleared after interrupt acknowledge.
The interrupt from INT1 can trigger interrupt as well as wakes up CPU from power-down mode.

ET0 : Timer 0 interrupt enable bit
0 := Disable TIMER0 interrupt
1 := Enable TIMER0 interrupt.

EX0 : External interrupt 0 enable bit
0 := Disable INT0 interrupt
1 := Enable INT0 interrupt.
A Negative-Edge from INT0 pin will trigger an interrupt if IT0(TCON.0) is set, and both of Negative-Edge and Positive-Edge will trigger an interrupt if IT0(TCON.0) is cleared. The interrupt flag IE0(TCON.1) will automatically cleared after interrupt acknowledge.
The interrupt from INT0 can trigger interrupt as well as wakes up CPU from power-down mode.

IP: Interrupt Priority Register

(MSB)							(LSB)
-	PLVD	PADC	-	PT1	PX1	PT0	PX0

Priority bit = 1 assigns high priority .
Priority bit = 0 assigns low priority.

Symbol	Position	Function
PLVD	IP.6	Low voltage detection interrupt priority.
PADC	IP.5	ADC interrupt priority bit.
PT1	IP.3	Timer 1 interrupt priority bit
PX1	IP.2	External interrupt 1 priority bit
PT0	IP.1	Timer 0 interrupt priority bit
PX0	IP.0	External interrupt 0 priority bit

TCON register: Timer/Counter Control Register

(MSB)							(LSB)
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Symbol	Position	Name and Significance
TF1	TCON.7	Timer 1 overflow Flag. Set by hardware on Timer/Counter overflow. cleared by hardware when processor vectors to interrupt routine.
TR1	TCON.6	Timer 1 Run control bit. Set/cleared by software to turn Timer/Counter on/off.
TF0	TCON.5	Timer 0 overflow Flag. Set by hardware on Timer/Counter overflow. cleared by hardware when processor vectors to interrupt routine.
TR0	TCON.4	Timer 0 Run control bit. Set/cleared by software to turn Timer/Counter on/off.
IE1	TCON.3	Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected.Cleared when interrupt processed.
IT1	TCON.2	Intenupt 1 Type control bit. Set/ cleared by software to specify falling edge/low level triggered external interrupts.
IE0	TCON.1	Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected.Cleared when interrupt processed.
IT0	TCON.0	Intenupt 0 Type control bit. Set/ cleared by software to specify falling edge/low level triggered external interrupts.

PCON: Power Control register

								LSB	
bit	Address	7	6	5	4	3	2	1	0
name	87H	-	-	LVDF	-	-	-	PD	IDL

LVDF : Low-Voltage Flag. Once low voltage condition is detected (VCC power is lower than LVD voltage), it is set by hardware (and should be cleared by software).
PD : Power-Down bit.
IDL : Idle mode bit.

INT_CLKO register

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	8FH	-	EX4	EX3	EX2	-	-	T1CKLO	T0CKLO

EX4 : External interrupt 4 enable bit
0 : = Disable INT4 interrupt
1 : = Enable INT4 interrupt.
Only Negatie-Edge from INT4 pin will trigger an interrupt to the CPU. The interrupt flag is implied, not user acceptable. The interrupt flag will be cleared after interrupt acknowledge or EX4 goes low.
The interrupt from INT4 can trigger interrupt as well as wakes up CPU from power-down mode.

EX3 : External interrupt 3 enable bit
0 : = Disable INT3 interrupt
1 : = Enable INT3 interrupt.
Only Negatie-Edge from INT3 pin will trigger an interrupt to the CPU. The interrupt flag is implied, not user acceptable. The interrupt flag will be cleared after interrupt acknowledge or EX3 goes low.
The interrupt from INT3 can trigger interrupt as well as wakes up CPU from power-down mode.

EX2 : External interrupt 2 enable bit
0 := Disable INT2 interrupt
1 := Enable INT2 interrupt.
Only Negative-Edge from INT2 pin will trigger an interrupt to the CPU. The interrupt flag is implied, not user acceptable. The interrupt flag will be cleared after interrupt acknowledge or EX2 goes low.
The interrupt from INT2 can trigger interrupt as well as wakes up CPU from power-down mode.

T1CKLO : When set, P3.4 is enabled to be the clock output of Timer 1. The clock rate is Timer 1 overflow rate divided by 2.
T0CKLO : When set, P3.5 is enabled to be the clock output of Timer 0. The clock rate is Timer 0 overflow rate divided by 2.

ADC_CONTR: AD Control register

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	BCH	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0

ADC_POWER(ADC_CONTR.7) : When clear, shut down the power of ADC bolck. When set, turn on the power of ADC block.
ADC_FLAG(ADC_CONTR.4) : ADC interrupt flag.

6.3 Interrupt Priorities

All interrupt sources, except INT2,INT3 and INT4, can also be individually programmed to one of two priority levels by setting or clearing the bits in Special Function Register IP. A low-priority interrupt can itself be interrupted by a high-pority interrupt, but not by another low-priority interrupt. A high-priority interrupt can't be interrupted by any other interrupt source.

If two requests of different priority levels are received simultaneously, the request of higher priority level is serviced. If requests of the same priority level are received simultaneously, an internal polling sequence determines which request is serviced. Thus within each priority level there is a second priority structure determined by the polling sequence,as follows:

	Source	Priority Within Level
0.	INT0	(highest)
1.	Timer 0	
2.	INT1	
3.	Timer 1	
5.	ADC interrupt	
6.	LVD	
7.	/INT2	
8.	/INT3	
9.	/INT4	(lowest)

Note that the "priority within level" structure is only used to resolve *simultaneous requests of the same prionty level.*

6.4 How Interrupts Are Handled

External interrupt pins and other interrupt sources are sampled at the rising edge of each instruction *OPcode fetch cycle.* The samples are polled during the next instruction *OPcode fetch cycle*. If one of the flags was in a set condition of the first cycle, the second cycle of polling cycles will find it and the interrupt system will generate an hardware LCALL to the appropriate service routine as long as it is not blocked by any of the following conditions.

Block conditions :

- An interrupt of equal or higher priority level is already in progress.
- The current cycle(polling cycle) is not the final cycle in the execution of the instruction in progress.
- The instruction in progress is RETI or any write to the IE, IP registers.
- The ISP/IAP activity is in progress.

Any of these four conditions will block the generation of the hardware LCALL to the interrupt service routine. Condition 2 ensures that the instruction in progress will be completed before vectoring into any service routine. Condition 3 ensures that if the instruction in progress is RETI or any access to IE, IP, then at least one or more instruction will be executed before any interrupt is vectored to.

The polling cycle is repeated with the last clock cycle of each instruction cycle. Note that if an interrupt flag is active but not being responded to for one of the above conditions, if the flag is not still active when the blocking condition is removed, the denied interrupt will not be serviced. In other words, the fact that the interrupt flag was once active but not being responded to for one of the above conditions, if the flag is not still active when the blocking condition is removed, the denied interrupt will not be serviced. The interrupt flag was once active but not serviced is not kept in memory. Every polling cycle is new.

Note that if an interrupt of higher priority level goes active prior to the rising edge of the third machine cycle, then in accordance with the above rules it will be vectored to during fifth and sixth machine cycle, without any instruction of the lower priority routine having been executed.

Thus the processor acknowledges an interrupt request by executing a hardware-generated LCALL to the appropriate servicing routine. In some cases it also clears the flag that generated the interrupt, and in other cases it doesn't. This has to be done in the user's software. The hardware-generated LCALL pushes the contents of the Program Counter onto the stack (but it does not save the PSW) and reloads the PC with an address that depends on the source of the interrupt being vectored to, as shown be low.

Source	Vector Address
External Interrupt 0	0003H
Timer 0	000BH
External Interrupt 1	0013H
Timer 1	001BH
ADC interrupt	002BH
LVD	0033H
External Interrupt 2	0053H
External Interrupt 3	005BH
External Interrupt 4	0083H

Execution proceeds from that location until the RETI instruction is encountered. The RETI instruction informs the processor that this interrupt routine is no longer in progress, then pops the top two bytes from the stack and reloads the Program Counter. Execution of the interrupted program continues from where it left off.

Note that a simple RET instruction would also have returned execution to the interrupted program, but it would have left the interrupt control system thinking an interrupt was still in progress.

6.5 External Interrupts

The external interrupt 0 and 1 can be programmed to be negative-edge-activated or both negative-edge-activated and positive-edge-activated by setting or clearing bit IT1 or IT0 in Register TCON. If ITx (x=0 or 1) is set, the external interrupts INTx (x=0 or 1) will be negative-edge-activated. In this mode if successive samples INTx(x=0,1) of the pin show a high in one cycle and a low in the next cycle, interrupt request flag IEx(x=0,1) in TCON is set. Flag bit IEx then requests the interrupt. If ITx (x=0 or 1) is cleared, the external interrupt INTx(x=0 or 1) will be triggered by either of Negative-Edge and Positive-Edge. In this mode if successive samples INTx(x=0,1) of the pin show a high in one cycle and a low in the next cycle or a low in one cycle and a high in the next cycle, interrupt request flag IEx in TCON is set and then requests the interrupt.

The External Interrupts $\overline{\text{INT2}}$ ~ $\overline{\text{INT4}}$ only can be negative-edge-activated. The interrupt flag is implied, not user acceptable. The interrupt flag will be cleared after interrupt acknowledge or EXn (n=2,3,4) in INT_CLKO register goes low.

All external interrupts can trigger interrupt as well as wakes up CPU from power-down mode.

Since the external interrupt pins are sampled once each machine cycle, an input high or low should hold for at least 12 system clocks to ensure sampling. In the external interrupt is transition-activated, the external source has to hold the request pin high for at least one machine cycle, and then hold it low for at least one machine cycle to ensure that the transition is seen so that interrupt request flag IEx will be set. IEx will be automatically cleared by the CPU when the service routine is called.

Chapter 7 Timer/Counter 0 and 1

Timer 0 and timer 1 are almost like the ones in the conventional 8051, both of them can be individually configured as timers or event counters.

In the “Timer” function, the register is incremented every 12 system clocks or every system clock depending on AUXR.7(T0x12) bit and AUXR.6(T1x12). In the default state, it is fully the same as the conventional 8051. In the x12 mode, the count rate equals to the system clock.

In the “Counter” function, the register is incremented in response to a 1-to-0 transition at its corresponding external input pin, T0 or T1. In this function, the external input is sampled once at the positive edge of every clock cycle. When the samples show a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the register during at the end of the cycle following the one in which the transition was detected. Since it takes 2 machine cycles (24 system clocks) to recognize a l-to-0 transition, the maximum count rate is 1/24 of the system clock. There are no restrictions on the duty cycle of the external input signal, but to ensure that a given level is sampled at least once before it changes, it should be held for at least one full machine cycle.

In addition to the “Timer” or “Counter” selection, Timer 0 and Timer 1 have four operating modes from which to select. The “Timer” or “Counter” function is selected by control bits C/$\overline{T}$ in the Special Function Register TMOD. These two Timer/Counter have four operating modes, which are selected by bit-pairs (M1, M0) in TMOD. Modes 0, 1, and 2 are the same for both Timer/Counter 0 and 1. Mode 3 is different.The four operating modes are described in the following text.

Symbol	Description	Address	Bit Address and Symbol MSB LSB								Value after Power-on or Reset
TCON	Timer Control	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000 0000B
TMOD	Timer Mode	89H	GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0	0000 0000B
TL0	Timer Low 0	8AH									0000 0000B
TL1	Timer Low 1	8BH									0000 0000B
TH0	Timer High 0	8CH									0000 0000B
TH1	Timer High 1	8DH									0000 0000B
AUXR	Auxiliary register	8EH	T0x12	T1x12	-	-	-	-	-	-	00xx xxxxB
INT_CLKO	External interrupt enable and Clock Output register	8FH	-	EX4	EX3	EX2	-	-	T1CLKO	T0CLKO	x000 xx00B

TCON register: Timer/Counter Control Register

(MSB) (LSB)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Symbol	Position	Name and Significance	Symbol	Position	Name and Significance
TF1	TCON.7	Timer 1 overflow Flag. Set by hardware on Timer/Counter overflow. cleared by hardware when processor vectors to interrupt routine.	IE1	TCON.3	Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected.Cleared when interrupt processed.
TR1	TCON.6	Timer 1 Run control bit. Set/cleared by software to turn Timer/Counter on/off.	IT1	TCON.2	Intenupt 1 Type control bit. Set/ cleared by software to specify falling edge/low level triggered external interrupts.
TF0	TCON.5	Timer 0 overflow Flag. Set by hardware on Timer/Counter overflow. cleared by hardware when processor vectors to interrupt routine.	IE0	TCON.1	Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected.Cleared when interrupt processed.
TR0	TCON.4	Timer 0 Run control bit. Set/cleared by software to turn Timer/Counter on/off.	IT0	TCON.0	Intenupt 0 Type control bit. Set/ cleared by software to specify falling edge/low level triggered external interrupts.

TMOD register : Timer/Counter Mode Control Register

(MSB) (LSB)

GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
Timer 1				Timer 0			

GATE Gating control when set.

C/$\overline{T}$ Timer or Counter Selector cleared for Timer operation (input from internal system clock). Set for Counter operation (input from "Tx"(x=0,1) input pin).

M1	M0	Operating Mode
0	0	16-bit auto-reload Timer/Counter for Timer 0 and Timer 1
0	1	16-bit Timer/Counter"THx"and"TLx"are cascaded;there is no prescaler
1	0	8-bit auto-reload Timer/Counter “THx” holds a value which is to be reloaded into “TLx” each time it overflows.
1	1	(Timer 0) TL0 is an 8-bit Timer/Counter controlled by the standard Timer 0 control bits TH0 is an 8-bit timer only controlled by Timer 1 control bits.
1	1	(Timer 1) Timer/Counter 1 stopped

AUXR register

LSB

bit	B7	B6	B5	B4	B3	B2	B1	B0
name	T0x12	T1x12	-	-	-	-	-	-

T0x12

0 : The clock source of Timer 0 is SYSclk/12.

1 : The clock source of Timer 0 is SYSclk/1.

T1x12

0 : The clock source of Timer 1 is SYSclk/12.

1 : The clock source of Timer 1 is SYSclk/1.

INT_CLKO : External interrupt enable register

B7	B6	B5	B4	B3	B2	B1	B0
-	EX4	EX3	EX2	-	-	T1CLKO	T0CLKO

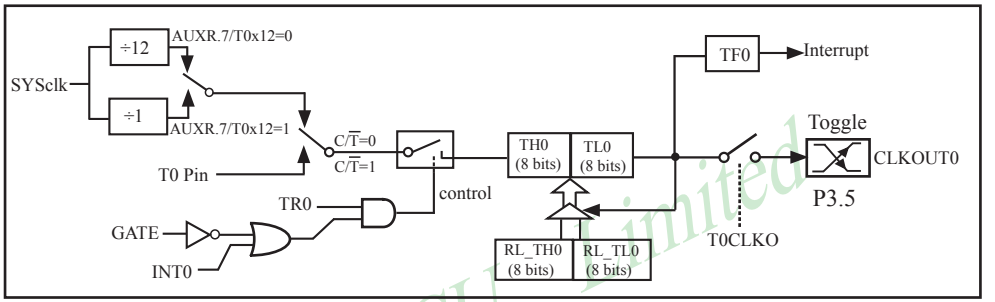
T1CLKO : When set, P3.4 is enabled to be the clock output of Timer 1. The clock rate is Timer 1 overflow rate divided by 2.

T0CLKO : When set, P3.5 is enabled to be the clock output of Timer 0. The clock rate is Timer 0 overflow rate divided by 2.

7.1 Timer/Counter 0 Mode of Operation

Mode 0

In this mode, the timer 0 is configured as a 16-bit re-loadable timer/counter. As the count rolls over from all 1s to all 0s, it sets the timer interrupt flag TF0. The counted input is enabled to the timer when TR0 = 1 and either GATE=0 or INT0= 1.(Setting GATE = 1 allows the Timer to be controlled by external input INT0, to facilitate pulse width measurements.) TR0 is a control bit in the Special Function Register TCON. GATE is in TMOD.



Timer/Counter 0 Mode 0: 16-Bit Auto-Reload

For Timer 0, there are 2 implied registers RL_TL0 and RL_TH0 implemented to meet Mode 0 operation requirement. The addressed of RL_TL0/RL_TH0 are homogeneous to TL0/TH0.

While the Timer 0 is configured to operate under Mode 0([M1 M0]TMOD[1:0]==00b), a write to TL0[7:0] will simultaneously write to RL_TL0 while TR0==0, but only write to RL_TL0 while TR0=1. A write to TH0[7:0] will simultaneously write to RL_TH0 while TR0==0, but only write to RL_TH0 while TR0=1.

Under MODE0 operating, overflow of {TH0,TL0} will automatically reload value {RL_TH0,RL_TL0} onto {TH0,TL0}.

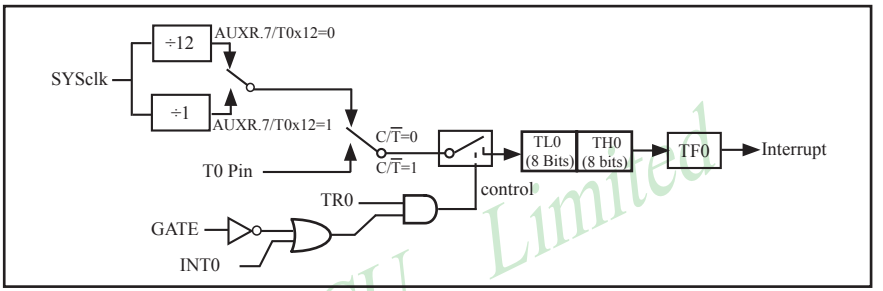
STC15-28-PIN is able to generate a programmable clock output on P3.5. When T0CLKO bit in INT_CLKO SFR is set, T0 timer overflow pulse will toggle P3.5 latch to generate a 50% duty clock. The frequency of clock-out is as following :

(SYSclk/2) / (256 – TH0), when T0x12=1
or **(SYSclk/2/12) / (256 – TH0) , when T0x12=0**

Mode 1

In this mode, the timer register is configured as a 16-bit register. As the count rolls over from all 1s to all 0s, it sets the timer interrupt flag TF0. The counted input is enabled to the timer when TR0 = 1 and either GATE=0 or INT0 = 1.(Setting GATE = 1 allows the Timer to be controlled by external input INT0, to facilitate pulse width measurements.) TR0 is a control bit in the Special Function Register TCON. GATE is in TMOD.

The 16-Bit register consists of all 8 bits of TH0 and the lower 8 bits of TL0. Setting the run flag (TR0) does not clear the registers.



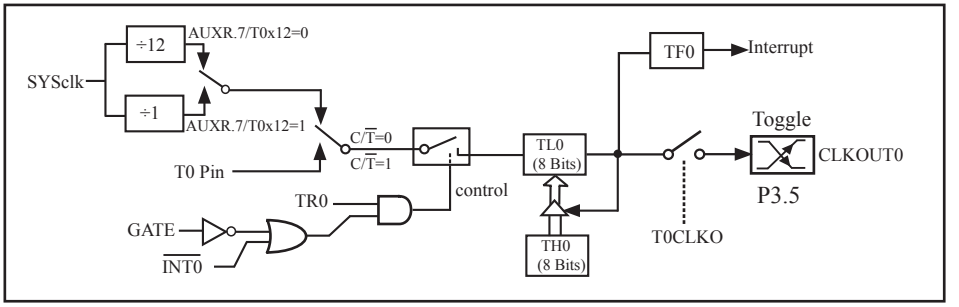
Timer/Counter 0 Mode 1 : 16-Bit Timer/Counter

Mode 2

Mode 2 configures the timer register as an 8-bit counter(TL0) with automatic reload. Overflow from TL0 not only set TF0, but also reload TL0 with the content of TH0, which is preset by software. The reload leaves TH0 unchanged.

STC15-28-PIN is able to generate a programmable clock output on P3.5. When T0CLKO bit in INT_CLKO SFR is set, T0 timer overflow pulse will toggle P3.5 latch to generate a 50% duty clock. The frequency of clock-out is as following :

(SYSclk/2) / (256 – TH0), when T0x12=1
or **(SYSclk/2/12) / (256 – TH0) , when T0x12=0**



Timer/Counter 0 Mode 2: 8-Bit Auto-Reload

Example: write a program using Timer 0 to create a 5KHz square wave on P1.0.

Assembly Language Solution:

```
        ORG     0030H
        MOV     TMOD,   #20H            ;8-bit auto-reload mode
        MOV     TL0,    #9CH            ;initialize TL0
        MOV     TH0,    #9CH            ;-100 reload value in TH0
        SETB    TR0                     ;Start Tmier 0
LOOP:   JNB     TF0,    LOOP            ;Wait for overflow
        CLR     TF0                     ;Clear Timer overflow flag
        CPL     P1.0                    ;Toggle port bit
        SJMP    LOOP                    ;Repeat
        END
```

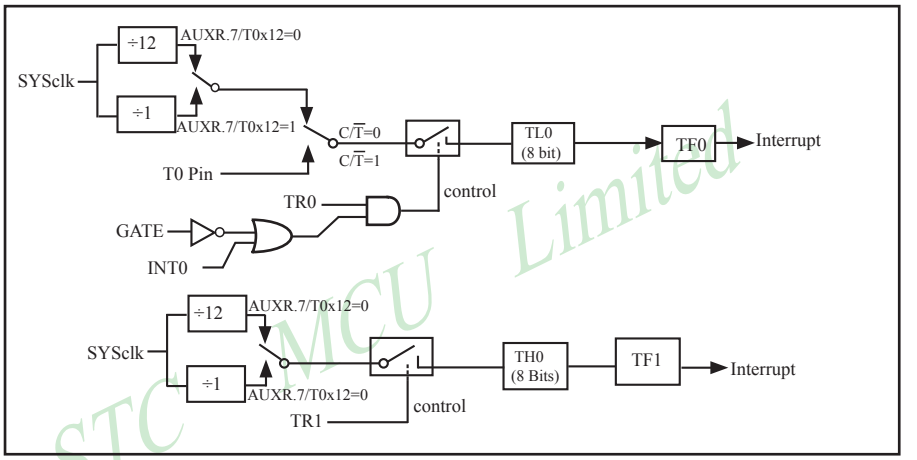
C Language Solution using Timer Interrupt :

```
#include <REG51.H>                      /* SFR declarations */
sbit    portbit = P1^0;                 /* Use variable portbit to refer to P1.0 */
main( )
{
        TMOD = 0x02;                    /* timer 0, mode 2 */
        TH0 = 9CH;                      /* 100us delay */
        TR0 = 1;                        /* Start timer */
        IE = 0x82                       /* Enable timer 0 interrupt */
        while(1);                       /* repeat forever */
}

void T0ISR(void) interrupt 1
{
        portbit = !portbit;             /*toggle port bit P1.0 */
}
```

Mode 3

Timer 1 in Mode 3 simply holds its count, the effect is the same as setting TR1 = 0. Timer 0 in Mode 3 established TL0 and TH0 as two separate 8-bit counters. TL0 use the Timer 0 control bits: $C/\overline{T}$,GATE,TR0, $\overline{INT0}$ and TF0. TH0 is locked into a timer function (counting machine cycles) and takes over the use of TR1 from Tmer 1. Thus, TH0 now controls the "Timer 1" interrupt.

Mode 3 is provided for applications requiring an extra 8-bit timer or counter. When Timer 0 is in Mode 3, Timer 1 can be turned on and off by switching it out of and into its own Mode 3, or can still be used by the serial port as a baud rate generator, or in fact, in any application not requiring an interrupt.

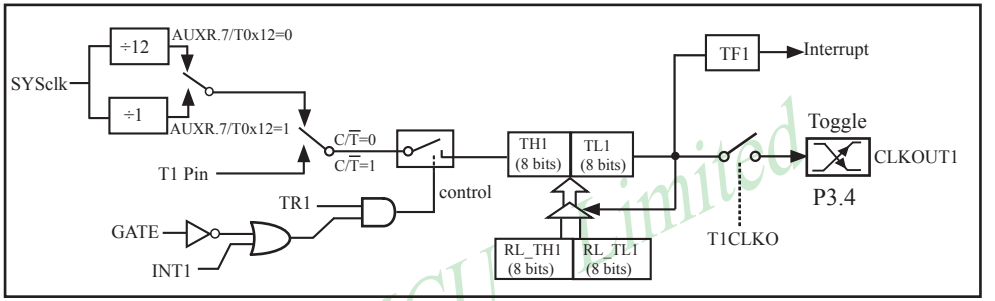


Timer/Counter 0 Mode 3: Two 8-Bit Counters

7.2 Timer/Counter 1 Mode of Operation

Mode 0

In this mode, the timer register is configured as a 16-bit re-loadable timer/counter. As the count rolls over from all 1s to all 0s, it sets the timer interrupt flag TF1. The counted input is enabled to the timer when TR1 = 1 and either GATE=0 or INT1 = 1. (Setting GATE = 1 allows the Timer to be controlled by external input INT1, to facilitate pulse width measurements.) TR0 is a control bit in the Special Function Register TCON. GATE is in TMOD.



Timer/Counter 1 Mode 0: 16-Bit Auto-Reload

For Timer 1, there are 2 implied registers RL_TL1 and RL_TH1 implemented to meet Mode 0 operation requirement. The addressed of RL_TL1/RL_TH1 are homogeneous to TL1/TH1.

While the Timer 1 is configured to operate under Mode 0([M1 M0]TMOD[5:4]==00b), a write to TL1[7:0] will simultaneously write to RL_TL1 while TR1==0, but only write to RL_TL1 while TR1=1. A write to TH1[7:0] will simultaneously write to RL_TH1 while TR1==0, but only write to RL_TH1 while TR1=1.

Under MODE0 operating, overflow of {TH1,TL1} will automatically reload value {RL_TH1,RL_TL1} onto {TH1,TL1}.

STC15-28-PIN is able to generate a programmable clock output on P3.4. When T1CLKO bit in INT_CLKO SFR is set, T1 timer overflow pulse will toggle P3.4 latch to generate a 50% duty clock. The frequency of clock-out is as following :

(SYSclk/2) / (256 – TH0), when T0x12=1
or **(SYSclk/2/12) / (256 – TH0) , when T0x12=0**

The following program is an assembly language code that domestrates 16-bit auto-reload Timer mode.

```
;/*---------------------------------------------------------------*/
;/* --- STC MCU International Limited ----------------------------*/
;/* --- STC 15 Series 16-bit auto-reload Timer Demo ---------------*/
;/* --- Mobile: (86)13922805190 ---------- ----------------------*/
;/* --- Fax: 86-755-82944243 -------------------------------------*/
;/* --- Tel: 86-755-82948412 -------------------------------------*/
;/* --- Web: www.STCMCU.com --------------------------------------*/
;/* If you want to use the program or the program referenced in the  */
;/* article, please specify in which data and procedures from STC    */
;/*---------------------------------------------------------------*/

;/* define constants */
#define MODE1T                  ;Timer clock mode,comment this line is 12T mode,uncomment is 1T mode

#ifdef MODE1T
T1MS    EQU 0B800H              ;1ms timer calculation method in 1T mode is (65536-18432000/1000)
#else
T1MS    EQU 0FA00H              ;1ms timer calculation method in 12T mode is (65536-18432000/12/1000)
#endif

;/* define SFR */
AUXR    DATA 8EH                ;Auxiliary register
TEST_LED  BIT  P1.0             ;work LED, flash once per second

;/* define variables */
COUNT DATA 20H                  ;1000 times counter (2 bytes)

;------------------------------------------------
  ORG   0000H
  LJMP  MAIN
  ORG   000BH
  LJMP  TM0_ISR

;------------------------------------------------
;/* main program */
MAIN:
#ifdef MODE1T
        MOV   AUXR,#80H         ;timer0 work in 1T mode
#endif
        MOV   TMOD,#00H         ;set timer0 as mode0 (16-bit auto-reload)
        MOV   TL0,#LOW T1MS     ;initial timer0 low byte
        MOV   TH0,#HIGH T1MS    ;initial timer0 high byte
```

```
        SETB    TR0                 ;timer0 start running
        SETB    ET0                 ;enable timer0 interrupt
        SETB    EA                  ;open global interrupt switch
        CLR     A
        MOV     COUNT,A
        MOV     COUNT+1,A           ;initial counter
        SJMP    $

;-------------------------------------------------

;/* Timer0 interrupt routine */
TM0_ISR:
        PUSH    ACC
        PUSH    PSW
        MOV     A,COUNT
        ORL     A,COUNT+1           ;check whether count(2byte) is equal to 0
        JNZ     SKIP
        MOV     COUNT,#LOW 1000     ;1ms * 1000 -> 1s
        MOV     COUNT+1,#HIGH 1000
        CPL     TEST_LED            ;work LED flash
SKIP:
        CLR     C
        MOV     A,COUNT             ;count--
        SUBB    A,#1
MOV     COUNT,A
        MOV     A,COUNT+1
        SUBB    A,#0
        MOV     COUNT+1,A
        POP     PSW
        POP     ACC
        RETI

;-------------------------------------------------

        END
```

The following program is an C language code that domestrates 16-bit auto-reload Timer mode.

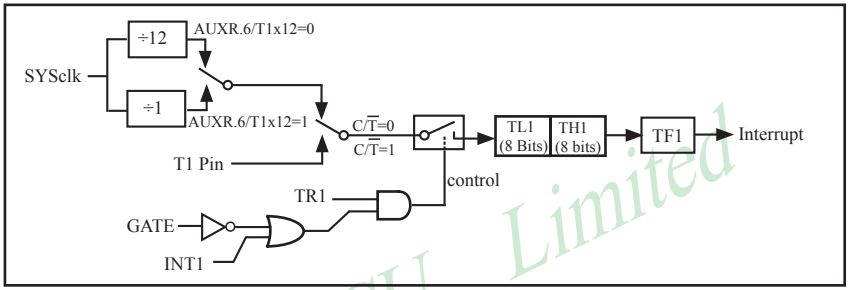
```
/*---------------------------------------------------------------*/
/* --- STC MCU International Limited ----------------------------*/
/* --- STC 15 Series 16-bit auto-reload Timer Demo ---------------*/
/* --- Mobile: (86)13922805190 ---------- ---------------------*/
/* --- Fax: 86-755-82944243 -------------------------------------*/
/* --- Tel: 86-755-82948412 -------------------------------------*/
/* --- Web: www.STCMCU.com --------------------------------------*/
/* If you want to use the program or the program referenced in the  */
/* article, please specify in which data and procedures from STC    */
/*---------------------------------------------------------------*/

#include "reg51.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

//-----------------------------------------------
/* define constants */
#define FOSC 18432000L
#define MODE1T                  //Timer clock mode, comment this line is 12T mode, uncomment is 1T mode

#ifdef MODE1T
#define T1MS (65536-FOSC/1000)      //1ms timer calculation method in 1T mode
#else
#define T1MS (65536-FOSC/12/1000)   //1ms timer calculation method in 12T mode
#endif

/* define SFR */
sfr AUXR = 0x8e;                    //Auxiliary register
sbit TEST_LED = P1^0;               //work LED, flash once per second

/* define variables */
WORD count;                         //1000 times counter

//-----------------------------------------------
/* Timer0 interrupt routine */
void tm0_isr() interrupt 1 using 1
{
    if (count-- == 0)               //1ms * 1000 -> 1s
    {
        count = 1000;               //reset counter
        TEST_LED = ! TEST_LED;      //work LED flash
    }
}
```

```
//-----------------------------------------------

/* main program */
void main()
{
#ifdef MODE1T
    AUXR = 0x80;                //timer0 work in 1T mode
#endif
    TMOD = 0x00;                //set timer0 as mode0 (16-bit auto-reload)
    TL0 = T1MS;                 //initial timer0 low byte
    TH0 = T1MS >> 8;            //initial timer0 high byte
    TR0 = 1;                    //timer0 start running
    ET0 = 1;                    //enable timer0 interrupt
    EA = 1;                     //open global interrupt switch
    count = 0;                  //initial counter

    while (1);                  //loop
}
```

Mode 1

In this mode, the timer register is configured as a 16-bit register. As the count rolls over from all 1s to all 0s, it sets the timer interrupt flag TF1. The counted input is enabled to the timer when TR1 = 1 and either GATE=0 or INT1= 1.(Setting GATE = 1 allows the Timer to be controlled by external input INT1, to facilitate pulse width measurements.) TRl is a control bit in the Special Function Register TCON. GATE is in TMOD.

The 16-Bit register consists of all 8 bits of THl and the lower 8 bits of TL1. Setting the run flag (TR1) does not clear the registers.



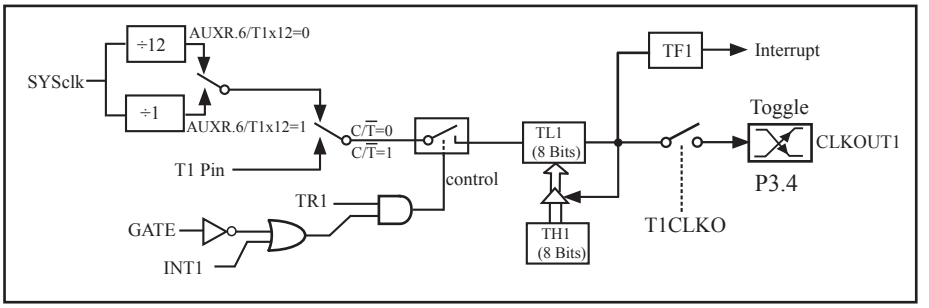
Timer/Counter 1 Mode 1 : 16-Bit Counter

Mode 2

Mode 2 configures the timer register as an 8-bit counter(TL1) with automatic reload. Overflow from TL1 not only set TFx, but also reload TL1 with the content of TH1, which is preset by software. The reload leaves TH1 unchanged.

STC15-28-PIN is able to generate a programmable clock output on P3.4. When T1CLKO bit in INT_CLKO SFR is set, T1 timer overflow pulse will toggle P3.4 latch to generate a 50% duty clock. The frequency of clock-out is as following :

(SYSclk/2) / (256 – TH1), when T1x12=1
or **(SYSclk/2/12) / (256 – TH1) , when T1x12=0**



Timer/Counter 1 Mode 2: 8-Bit Auto-Reload

7.3 Changes of STC15-28-PIN Timers compared with standard 8051

The TIMER0 and Timer1 are almost the same to standard 80C51 MCU excepting the following changes.

TIMER0 and TIMER1 Clock Sources

SFR Name	SFR Address	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	97H	T0X12	T1X12	-	-	-	-	-	-

T0X12
0 := The clock source of Timer 0 is SYSclk/12.
1 := The clock source of Timer 0 is SYSclk.

T1X12
0 := The clock source of Timer 1 is SYSclk/12.
1 := The clock source of Timer 1 is SYSclk.

Change MODE0 functionality

The MODE0 operations for TIMER1 and TIMER0 have been changed to 16-bit re-loadable timer/counter from 13-bit timer/counter.

There are 4 implied registers RL_TL0, RL_TH0, RL_TL1, and RL_TH1 implemented to meet MODE0 operation requirement. The addressed of RL_TL0/RL_TH0/RL_TL1/RL_TH1 are homogeneous to TL0/TH0/TL1/TH1.

While the TIMER0 is configured to operate under MODE0(TMOD[1:0]==00b), a write to TL0[7:0] will simultaneously write to RL_TL0 while TR0==0, but only write to RL_TL0 while TR0=1. A write to TH0[7:0] will simultaneously write to RL_TH0 while TR0==0, but only write to RL_TH0 while TR0=1.

While the TIMER1 is configured to operate under MODE0(TMOD[5:4]==00b), a write to TL1[7:0] will simultaneously write to RL_TL1 while TR1==0, but only write to RL_TL1 while TR1=1. A write to TH1[7:0] will simultaneously write to RL_TH1 while TR1==0, but only write to RL_TH1 while TR1=1.

Under MODE0 operating, overflow of {TH0,TL0} will automatically reload value {RL_TH0,RL_TL0} onto {TH0,TL0}.

Under MODE0 operating, overflow of {TH0,TL0} will automatically reload value {RL_TH0,RL_TL0} onto {TH0,TL0}.

7.4 Generic Programmable Clock Output

There are 3 generic clocks can be induced to I/O pins.

SFR Name	SFR Address	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO	8FH	-	EX4	EX3	EX2	-	-	T1CLKO	T0CLKO

SFR Name	SFR Address	B7	B6	B5	B4	B3	B2	B1	B0
IRC_CLKO	BBH	EN_IRCO	-	-	-	DIVIRCO	-	-	-

Output Clock from system clock(Internal RC) to P0.0

Set EN_IRCO(IRC_CLKO.7) to switch P0.0 into IRC clock output pin. Depending on DIVIRCO set or clear, the output frequency will be SYSclk/2 or SYSclk.

Output Clock from TIMER0 Overflow onto P3.5

Setting T0CLKO can switch P3.5 into clock output pin, and the clock with frequency TIMER0-Overflow-Rate divided by 2. The frequency of clock-out is as following :

(SYSclk/2) / (256 – TH0), when T0x12=1

or **(SYSclk/2/12) / (256 – TH0) , when T0x12=0**

STC15-28-PIN Output Clock from TIMER1 Overflow onto P3.4

Setting T1CLKO can switch P3.4 into clock output pin, and the clock with frequency TIMER1-Overflow-Rate divided by 2. The frequency of clock-out is as following :

(SYSclk/2) / (256 – TH1), when T1x12=1

or **(SYSclk/2/12) / (256 – TH1) , when T1x12=0**

The following program is an assembly language code that domestrates Programmable Clock Output function.

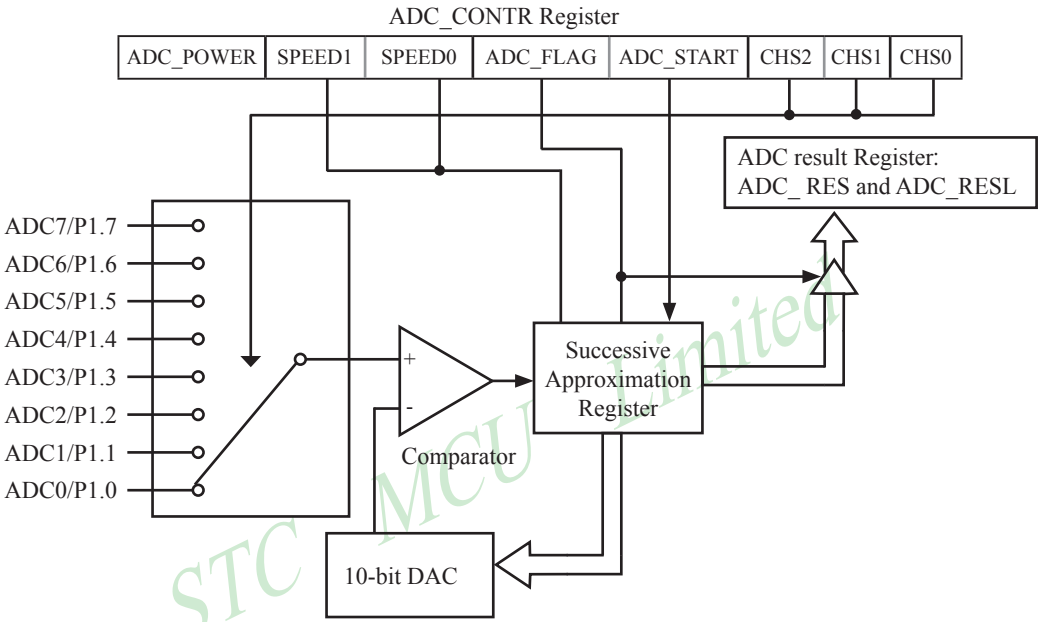
```
;/*---------------------------------------------------------------*/
;/* --- STC MCU International Limited ----------------------------*/
;/* --- STC 15 Series Programmable Clock Output Demo --------------*/
;/* --- Mobile: (86)13922805190 ---------- ----------------------*/
;/* --- Fax: 86-755-82944243 -------------------------------------*/
;/* --- Tel: 86-755-82948412 -------------------------------------*/
;/* --- Web: www.STCMCU.com --------------------------------------*/
;/* If you want to use the program or the program referenced in the  */
;/* article, please specify in which data and procedures from STC    */
;/*---------------------------------------------------------------*/

;/* define constants */
#define MODE1T                  ;Timer clock mode, comment this line is 12T mode, uncomment is 1T mode

#ifdef MODE1T
F38_4KHz    EQU 0FF10H      ;38.4KHz frequency calculation method of 1T mode is (65536-18432000/2/38400)
#else
F38_4KHz    EQU 0FFECH  ;38.4KHz frequency calculation method of 12T mode(65536-18432000/2/12/38400)
#endif

;/* define SFR */
AUXR        DATA    08EH            ;Auxiliary register
WAKE_CLKO   DATA    08FH            ;wakeup and clock output control register
T0CLKO      BIT     P3.5            ;timer0 clock output pin

;------------------------------------------------
    ORG 0000H
    LJMP    MAIN

;------------------------------------------------
;/* main program */
MAIN:
#ifdef MODE1T
    MOV     AUXR,#80H               ;timer0 work in 1T mode
#endif
    MOV     TMOD,#00H               ;set timer0 as mode0 (16-bit auto-reload)
    MOV     TL0,#LOW F38_4KHz       ;initial timer0 low byte
    MOV     TH0,#HIGH F38_4KHz      ;initial timer0 high byte
    SETB    TR0
    MOV     WAKE_CLKO,#01H          ;enable timer0 clock output

    SJMP    $

;------------------------------------------------
    END
```

The following program is an C language code that domestrates Programmable Clock Output function.

```
/*---------------------------------------------------------------------*/
/* --- STC MCU International Limited ---------------------------------*/
/* --- STC 15 Series Programmable Clock Output Demo ----------------*/
/* --- Mobile: (86)13922805190 ---------- ----------------------*/
/* --- Fax: 86-755-82944243 ------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------*/
/* If you want to use the program or the program referenced in the  */
/* article, please specify in which data and procedures from STC    */
/*---------------------------------------------------------------------*/

#include "reg51.h"

//-----------------------------------------------

/* define constants */
#define FOSC 18432000L
//#define MODE1T                //Timer clock mode, comment this line is 12T mode, uncomment is 1T mode

#ifdef MODE1T
#define F38_4KHz (65536-FOSC/2/38400)      //38.4KHz frequency calculation method of 1T mode
#else
#define F38_4KHz (65536-FOSC/2/12/38400)   //38.4KHz frequency calculation method of 12T mode
#endif

/* define SFR */
sfr AUXR      = 0x8e;                      //Auxiliary register
sfr WAKE_CLKO = 0x8f;                       //wakeup and clock output control register
sbit T0CLKO   = P3^5;                      //timer0 clock output pin

//-----------------------------------------------

/* main program */
void main()
{
#ifdef MODE1T
   AUXR = 0x80;                            //timer0 work in 1T mode
#endif
   TMOD = 0x00;                             //set timer0 as mode0 (16-bit auto-reload)
   TL0 = F38_4KHz;                         //initial timer0 low byte
   TH0 = F38_4KHz >> 8;                    //initial timer0 high byte
   TR0 = 1;                                //timer0 start running
   WAKE_CLKO = 0x01;                       //enable timer0 clock output

   while (1);                              //loop
}
```

Chapter 8 Analog to Digital Converter

8.1 A/D Converter Structure



The ADC on STC15-28-PIN is an 10-bit resolution, successive-approximation approach, medium-speed A/D converter.

Conversion is invoked since ADC_STRAT(ADC_CONTR.3) bit is set. Before invoking conversion, ADC_POWER/ADC_CONTR.7 bit should be set first in order to turn on the power of analog front-end in ADC circuitry. Prior to ADC conversion, the desired I/O ports for analog inputs should be configured as input-only or open-drain mode first. The converter takes around a fourth cycles to sample analog input data and other three fourths cycles in successive-approximation steps. Total conversion time is controlled by two register bits – SPEED1 and SPEED0. Eight analog channels are available on P1 and only one of them is connected to to the comparator depending on the selection bits {CHS2,CHS1,CHS0}. When conversion is completed, the result will be saved onto {ADC_RES,ADC_RESL[1:0]} register. After the result are completed and saved, ADC_FLAG is also set.ADC_FLAG associated with its enable register IE.5(EADC). ADC_FLAG should be cleared in software. The ADC interrupt service routine vectors to 2Bh . When the chip enters idle mode or power-down mode, the power of ADC is gated off by hardware.

$$10\text{Conversion Result:(ADC\_RES[7:0], ADC\_RESL[1:0]} = 1024 \times \frac{Vin}{Vcc}$$

V_{in} is the input voltage for analog channel, and V_{CC} is the MCU actual operating voltage whose referece voltage is MCU operating voltage.

8.2 Register for ADC

SFR Name	SFR Address	B7	B6	B5	B4	B3	B2	B1	B0
P1ASF	9DH	P17ASF	P16ASF	P15ASF	P14ASF	P13ASF	P12ASF	P11ASF	P10ASF

P1xASF
0 : = Keep P1.x as general-purpose I/O function.
1 : = Set P1.x as ADC input channel-x

ADC_CONTR(ADC Control register)

LSB

bit	B7	B6	B5	B4	B3	B2	B1	B0
name	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0

ADC_POWER : When clear shut down the power of ADC block. When set turn on the power of ADC block.
SPEED1, SPEED0 : Conversion speed selection.
00 : 540 clock cycles are needed for a conversion.
01 : 360 clock cycles are needed for a conversion.
10 : 180 clock cycles are needed for a conversion.
11 : 90 clock cycles are needed for a conversion.

ADC_FLAG : ADC interrupt flag.It will be set by the device after the device has finished a conversion, and should be cleared by the user's software.

ADC_STRAT : ADC start bit, which enable ADC conversion.It will automatically cleared by the device after the device has finished the conversion.

CHS2 ~ CHS0 : Used to select one analog input source from 8 channels.

CHS2	CHS1	CHS0	Source
0	0	0	P1.0 (default) as the A/D channel input
0	0	1	P1.1 as the A/D channel input
0	1	0	P1.2 as the A/D channel input
0	1	1	P1.3 as the A/D channel input
1	0	0	P1.4 as the A/D channel input
1	0	1	P1.5 as the A/D channel input
1	1	0	P1.6 as the A/D channel input
1	1	1	P1.7 as the A/D channel input

Note : The corresponding bits in P1ASF should be configured correctly before starting A/D conversion. The sepecificP1ASF bits should be set corresponding with the desired channels.

ADC_RES(ADC result register)

bit	B7	B6	B5	B4	B3	B2	B1	B0
name								

The ADC_RES is the final result from the A/D conversion

ADC_RESL(Low Byte of ADC result register)

bit	B7	B6	B5	B4	B3	B2	B1	B0
name								

IE: Interrupt Enable Rsgister

(MSB) (LSB)

EA	ELVD	EADC	-	ET1	EX1	ET0	EX0

Enable Bit = 1 enables the interrupt .
Enable Bit = 0 disables it .

Symbol	Position	Function
EA	IE.7	disables all interrupts. if EA = 0,no interrupt will be acknowledged. if EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
EADC	IE.5	ADC interrupt enable bit

IP: Interrupt Priority Register

(MSB) (LSB)

-	PLVD	PADC	-	PT1	PX1	PT0	PX0

Priority bit = 1 assigns high priority .
Priority bit = 0 assigns low priority.

PADC IP.5 ADC interrupt priority bit.

8.3 Program using interrupts to demostrate A/D Conversion

There are two example procedures using interrupts to demostrate A/D conversion, one written in assembly langugage and the other in C language.

Assembly language code listing:

```
;/*---------------------------------------------------------------------------*/
;/* --- STC MCU International Limited ---------------------------------------*/
;/* --- STC 1T Series MCU A/D Conversion Demo --------------------------------*/
;/* --- Mobile: (86)13922805190 ----------------------------------------------*/
;/* --- Fax: 86-755-82944243 -------------------------------------------------*/
;/* --- Tel: 86-755-82948412 -------------------------------------------------*/
;/* --- Web: www.STCMCU.com --------------------------------------------------*/
;/* If you want to use the program or the program referenced in the  */
;/* article, please specify in which data and procedures from STC    */
;/*---------------------------------------------------------------------------*/
 ;/*Declare SFR associated with the ADC */
                ADC_CONTR       EQU     0BCH            ;ADC control register
                ADC_RES         EQU     0BDH            ;ADC high 8-bit result register
                ADC_LOW2        EQU     0BEH            ;ADC low 2-bit result register
                P1ASF           EQU     09DH            ;P1 secondary function control register

;/*Define ADC operation const for ADC_CONTR*/
                ADC_POWER       EQU     80H             ;ADC power control bit
                ADC_FLAG        EQU     10H             ;ADC complete flag
                ADC_START       EQU     08H             ;ADC start control bit
                ADC_SPEEDLL     EQU     00H             ;540 clocks
                ADC_SPEEDL      EQU     20H             ;360 clocks
                ADC_SPEEDH      EQU     40H             ;180 clocks
                ADC_SPEEDHH     EQU     60H             ;90 clocks

                ADCCH           DATA    20H             ;ADC channel NO.
;------------------------------------------------------------------------------
                        ORG     0000H
                        LJMP    MAIN

                        ORG     002BH
                        LJMP    ADC_ISR
;------------------------------------------------------------------------------
                        ORG     0100H
        MAIN:
                        MOV     SP,     #3FH
                        MOV     ADCCH, #0
                        LCALL   INIT_UART               ;Init UART, use to show ADC result
                        LCALL   INIT_ADC                ;Init ADC sfr
                        MOV     IE,     #0A0H           ;Enable ADC interrupt
                                                        ;and Open master interrupt switch
                        SJMP    $
```

```
;/*---------------------------------------
;ADC interrupt service routine
;---------------------------------------*/
        ADC_ISR:
                        PUSH    ACC
                        PUSH    PSW
                        ANL     ADC_CONTR,      #NOT ADC_FLAG           ;Clear ADC interrupt flag
                        MOV     A,          ADCCH
                        LCALL   SEND_DATA                               ;Send channel NO.
                        MOV     A,          ADC_RES                     ;Get ADC high 8-bit result
                        LCALL   SEND_DATA                               ;Send to UART

;//if you want show 10-bit result, uncomment next 2 lines
;                       MOV     A,          ADC_LOW2                    ;Get ADC low 2-bit result
;                       LCALL   SEND_DATA                               ;Send to UART
                        INC     ADCCH
                        MOV     A,          ADCCH
                        ANL     A,          #07H
                        MOV     ADCCH, A
                        ORL     A,          #ADC_POWER | ADC_SPEEDLL | ADC_START
                        MOV     ADC_CONTR,      A                       ;ADC power-on delay
                                                                        ;and re-start A/D conversion
                        POP     PSW
                        POP     ACC
                        RETI
;/*---------------------------------
;Initial ADC sfr
;---------------------------------*/
        INIT_ADC:
                        MOV     P1ASF,  #0FFH                           ;Set all P1 as analog input port
                        MOV     ADC_RES,        #0                      ;Clear previous result
                        MOV     A,          ADCCH
                        ORL     A,          #ADC_POWER | ADC_SPEEDLL | ADC_START
                        MOV     ADC_CONTR,      A                       ;ADC power-on delay
                                                                        ;and Start A/D conversion
                        MOV     A,          #2
                        LCALL   DELAY
                        RET
;/*-----------------------------------
;Initial UART
;---------------------------------------*/
        INIT_UART:
                        MOV     SCON,   #5AH                            ;8 bit data ,no parity bit
                        MOV     TMOD,   #20H                            ;T1 as 8-bit auto reload
                        MOV     A,          #-5                 ;Set Uart baudrate -(18432000/12/32/9600)
                        MOV     TH1,        A                           ;Set T1 reload value
                        MOV     TL1,        A
                        SETB    TR1                                     ;T1 start running
                        RET
```

```
;/*---------------------------------------
;Send one byte data to PC
;Input: ACC (UART data)
;Output:-
;---------------------------------------*/
        SEND_DATA:
                        JNB     TI,     $       ;Wait for the previous data is sent
                        CLR     TI              ;Clear TI flag
                        MOV     SBUF,   A       ;Send current data
                        RET
;/*---------------------------------------
;Software delay function
;---------------------------------------*/
        DELAY:
                        MOV     R2,     A
                        CLR     A
                        MOV     R0,     A
                        MOV     R1,     A
        DELAY1:
                        DJNZ    R0,     DELAY1
                        DJNZ    R1,     DELAY1
                        DJNZ    R2,     DELAY1
                        RET

                        END
```

C language code listing:

```
/*-------------------------------------------------------------------------------*/
/* --- STC MCU International Limited ---------------------------------------*/
/* --- STC 1T Series MCU A/D Conversion Demo -------------------------------*/
/* --- Mobile: (86)13922805190 ---------------------------------------------*/
/* --- Fax: 86-755-82944243 ------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------*/
/* If you want to use the program or the program referenced in the  */
/* article, please specify in which data and procedures from STC    */
/*-------------------------------------------------------------------------------*/

        #include "reg51.h"
        #include "intrins.h"

        #define FOSC    18432000L
        #define BAUD    9600

        typedef unsigned char BYTE;
        typedef unsigned int WORD;
```

```
/*Declare SFR associated with the ADC */
sfr     ADC_CONTR   =  0xBC;        //ADC control register
sfr     ADC_RES     =  0xBD;        //ADC hight 8-bit result register
sfr     ADC_LOW2    =  0xBE;        //ADC low 2-bit result register
sfr     P1ASF       =  0x9D;        //P1 secondary function control register

/*Define ADC operation const for ADC_CONTR*/
#define  ADC_POWER    0x80           //ADC power control bit
#define  ADC_FLAG     0x10           //ADC complete flag
#define  ADC_START    0x08           //ADC start control bit
#define  ADC_SPEEDLL  0x00           //540 clocks
#define  ADC_SPEEDL   0x20           //360 clocks
#define  ADC_SPEEDH   0x40           //180 clocks
#define  ADC_SPEEDHH  0x60           //90 clocks

        void InitUart();
        void SendData(BYTE dat);
        void Delay(WORD n);
        void InitADC();
        BYTE ch = 0;                //ADC channel NO.

void main()
{
        InitUart();                 //Init UART, use to show ADC result
        InitADC();                  //Init ADC sfr
        IE = 0xa0;                  //Enable ADC interrupt and Open master interrupt switch
                                    //Start A/D conversion
        while (1);
        }

/*-------------------------------------------
  ADC interrupt service routine
-------------------------------------------*/
void adc_isr( ) interrupt 5 using 1
{
        ADC_CONTR &= !ADC_FLAG;     //Clear ADC interrupt flag

        SendData(ch);               //Show Channel NO.
        SendData(ADC_RES);          //Get ADC high 8-bit result and Send to UART

        //if you want show 10-bit result, uncomment next line
        //    SendData(ADC_LOW2);   //Show ADC low 2-bit result

        if (++ch > 7) ch = 0;       //switch to next channel
        ADC_CONTR = ADC_POWER | ADC_SPEEDLL | ADC_START | ch;
}
```

```
/*--------------------------------
   Initial  ADC  sfr
----------------------------------------*/
void InitADC( )
{
        P1ASF = 0xff;                                   //Set all P1 as analog input port
        ADC_RES = 0;                                    //Clear previous result
        ADC_CONTR = ADC_POWER | ADC_SPEEDLL | ADC_START | ch;
        Delay(2);                                       //ADC power-on delay and Start A/D conversion
}
/*--------------------------------
    Initial UART
----------------------------------------*/
void InitUart()
{
        SCON = 0x5a;                                    //8 bit data ,no parity bit
        TMOD = 0x20;                                    //T1 as 8-bit auto reload
        TH1 = TL1 = -(FOSC/12/32/BAUD);                 //Set Uart baudrate
        TR1 = 1;                                        //T1 start running
 }
/*--------------------------------
 Send one byte data to PC
 Input: dat (UART data)
 Output:-
----------------------------------------*/
void SendData(BYTE dat)
{
        while (!TI);                                    //Wait for the previous data is sent
        TI = 0;                                         //Clear TI flag
        SBUF = dat;                                     //Send current data
 }
/*--------------------------------
 Software delay function
----------------------------------------*/
void Delay(WORD n)
{
        WORD x;

        while (n--)
        {
                x = 5000;
                while (x--);
         }
 }
```

8.4 Program using inquiry to demostrate A/D Conversion

There are two example procedures using inquiry to demostrate A/D conversion, one written in assembly langugage and the other in C language.

Assembly language code listing:

```
;/*---------------------------------------------------------------------------*/
;/* --- STC MCU International Limited ---------------------------------------*/
;/* --- STC 1T Series MCU A/D Conversion Demo --------------------------------*/
;/* --- Mobile: (86)13922805190 ----------------------------------------------*/
;/* --- Fax: 86-755-82944243 -------------------------------------------------*/
;/* --- Tel: 86-755-82948412 -------------------------------------------------*/
;/* --- Web: www.STCMCU.com --------------------------------------------------*/
;/* If you want to use the program or the program referenced in the  */
;/* article, please specify in which data and procedures from STC    */
;/*---------------------------------------------------------------------------*/
 ;/*Declare SFR associated with the ADC */
                ADC_CONTR       EQU     0BCH            ;ADC control register
                ADC_RES         EQU     0BDH            ;ADC high 8-bit result register
                ADC_LOW2        EQU     0BEH            ;ADC low 2-bit result register
                P1ASF           EQU     09DH            ;P1 secondary function control register
;/*Define ADC operation const for ADC_CONTR*/
                ADC_POWER       EQU     80H             ;ADC power control bit
                ADC_FLAG        EQU     10H             ;ADC complete flag
                ADC_START       EQU     08H             ;ADC start control bit
                ADC_SPEEDLL     EQU     00H             ;540 clocks
                ADC_SPEEDL      EQU     20H             ;360 clocks
                ADC_SPEEDH      EQU     40H             ;180 clocks
                ADC_SPEEDHH     EQU     60H             ;90 clocks

;------------------------------------------------------------------------------
                ORG     0000H
                LJMP    MAIN
;------------------------------------------------------------------------------
                ORG     0100H
MAIN:
                LCALL   INIT_UART                       ;Init UART, use to show ADC result
                LCALL   INIT_ADC                        ;Init ADC sfr
;------------------------------------------------------------------------------
NEXT:
                MOV     A,      #0
                LCALL   SHOW_RESULT                     ;Show channel0 result
                MOV     A,      #1
                LCALL   SHOW_RESULT                     ;Show channel1 result
                MOV     A,      #2
                LCALL   SHOW_RESULT                     ;Show channel2 result
                MOV     A,      #3
                LCALL   SHOW_RESULT                     ;Show channel3 result
```

```
                MOV     A,      #4
                LCALL   SHOW_RESULT             ;Show channel4 result
                MOV     A,      #5
                LCALL   SHOW_RESULT             ;Show channel5 result
                MOV     A,      #6
                LCALL   SHOW_RESULT             ;Show channel6 result
                MOV     A,      #7
                LCALL   SHOW_RESULT             ;Show channel7 result
                SJMP    NEXT
;/*-------------------------------------------------------------
;Send ADC result to UART
;Input: ACC (ADC channel NO.)
;Output:-
;-------------------------------------------------------------*/
SHOW_RESULT:
                LCALL   SEND_DATA               ;Show Channel NO.
                LCALL   GET_ADC_RESULT          ;Get high 8-bit ADC result
                LCALL   SEND_DATA               ;Show result
;//if you want show 10-bit result, uncomment next 2 lines
;               MOV     A,      ADC_LOW2        ;Get low 2-bit ADC result
;               LCALL   SEND_DATA               ;Show result
                RET
;/*-------------------------------------------------------------
;Read ADC conversion result
;Input: ACC (ADC channel NO.)
;Output:ACC (ADC result)
;-------------------------------------------------------------*/
GET_ADC_RESULT:
                ORL     A,      #ADC_POWER | ADC_SPEEDLL | ADC_START
                MOV     ADC_CONTR,A             ;Start A/D conversion
                NOP                             ;Must wait before inquiry
                NOP
                NOP
                NOP
WAIT:
                MOV     A,ADC_CONTR                             ;Wait complete flag
                JNB     ACC.4,  WAIT                            ;ADC_FLAG(ADC_CONTR.4)
                ANL     ADC_CONTR,      #NOT ADC_FLAG           ;Clear ADC_FLAG
                MOV     A,      ADC_RES                         ;Return ADC result
                RET
;/*-------------------------------------------------------------
;Initial ADC sfr
;-------------------------------------------------------------*/
INIT_ADC:
                MOV     P1ASF,  #0FFH                           ;Open 8 channels ADC function
                MOV     ADC_RES,        #0                      ;Clear previous result
                MOV     ADC_CONTR,      #ADC_POWER | ADC_SPEEDLL
                MOV     A,      #2                              ;ADC power-on and delay
                LCALL   DELAY
                RET
```

```
;/*-------------------------------------------------------
;Initial UART
;-------------------------------------------------------*/
INIT_UART:
                MOV     SCON,   #5AH            ;8 bit data ,no parity bit
                MOV     TMOD,   #20H            ;T1 as 8-bit auto reload
                MOV     A,      #-5             ;Set Uart baudrate -(18432000/12/32/9600)
                MOV     TH1,    A               ;Set T1 reload value
                MOV     TL1,    A
                SETB    TR1                     ;T1 start running
                RET
;/*-------------------------------------------------------
;Send one byte data to PC
;Input: ACC (UART data)
;Output:-
;-------------------------------------------------------*/
SEND_DATA:
                NB      TI,     $               ;Wait for the previous data is sent
                CLR     TI                      ;Clear TI flag
                MOV     SBUF,   A               ;Send current data
                RET
;/*----------------------------
;Software delay function
;----------------------------*/
DELAY:
                MOV     R2,     A
                CLR     A
                MOV     R0,     A
                MOV     R1,     A
DELAY1:
                DJNZ    R0,     DELAY1
                DJNZ    R1,     DELAY1
                DJNZ    R2,     DELAY1
                RET
                END
```


C language code listing:

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU International Limited -----------------------------------*/
/* --- STC 1T Series MCU A/D Conversion Demo ----------------------*/
/* --- Mobile: (86)13922805190 -----------------------------------------*/
/* --- Fax: 86-755-82944243 --------------------------------------------*/
/* --- Tel: 86-755-82948412 --------------------------------------------*/
/* --- Web: www.STCMCU.com ---------------------------------------------*/
/* If you want to use the program or the program referenced in the  */
/* article, please specify in which data and procedures from STC    */
/*---------------------------------------------------------------------------*/
```

```
#include "reg51.h"
#include "intrins.h"

#define FOSC    18432000L
#define BAUD    9600

typedef unsigned char BYTE;
typedef unsigned int WORD;

/*Declare SFR associated with the ADC */
sfr     ADC_CONTR   =   0xBC;       //ADC control register
sfr     ADC_RES     =   0xBD;       //ADC high 8-bit result register
sfr     ADC_LOW2    =   0xBE;       //ADC low 2-bit result register
sfr     P1ASF       =   0x9D;       //P1 secondary function control register

/*Define ADC operation const for ADC_CONTR*/
#define ADC_POWER   0x80            //ADC power control bit
#define ADC_FLAG    0x10            //ADC complete flag
#define ADC_START   0x08            //ADC start control bit
#define ADC_SPEEDLL 0x00            //540 clocks
#define ADC_SPEEDL  0x20            //360 clocks
#define ADC_SPEEDH  0x40            //180 clocks
#define ADC_SPEEDHH 0x60            //90 clocks

void InitUart();
void InitADC();
void SendData(BYTE dat);
BYTE GetADCResult(BYTE ch);
void Delay(WORD n);
void ShowResult(BYTE ch);

void main()
{
    InitUart();                     //Init UART, use to show ADC result
    InitADC();                      //Init ADC sfr
    while (1)
    {
        ShowResult(0);              //Show Channel0
        ShowResult(1);              //Show Channel1
        ShowResult(2);              //Show Channel2
        ShowResult(3);              //Show Channel3
        ShowResult(4);              //Show Channel4
        ShowResult(5);              //Show Channel5
        ShowResult(6);              //Show Channel6
        ShowResult(7);              //Show Channel7
    }
}
```

```
/*---------------------------------------------------
Send ADC result to UART
-----------------------------------------------------------------*/
void ShowResult(BYTE ch)
{
    SendData(ch);                       //Show Channel NO.
    SendData(GetADCResult(ch));         //Show ADC high 8-bit result

    //if you want show 10-bit result, uncomment next line
    //  SendData(ADC_LOW2);             //Show ADC low 2-bit result
}
/*---------------------------------------------------
Get ADC result
-----------------------------------------------------------------*/
BYTE GetADCResult(BYTE ch)
{
    ADC_CONTR = ADC_POWER | ADC_SPEEDLL | ch | ADC_START;
    _nop_();                            //Must wait before inquiry
    _nop_();
    _nop_();
    _nop_();
    while (!(ADC_CONTR & ADC_FLAG));    //Wait complete flag
    ADC_CONTR &= ~ADC_FLAG;             //Close ADC

    return ADC_RES;                     //Return ADC result
}

/*---------------------------------------------------
    Initial UART
-----------------------------------------------------------------*/
void InitUart()
{
    SCON = 0x5a;                        //8 bit data ,no parity bit
    TMOD = 0x20;                        //T1 as 8-bit auto reload
    TH1 = TL1 = -(FOSC/12/32/BAUD);     //Set Uart baudrate
    TR1 = 1;                            //T1 start running
}

/*---------------------------------------------------
    Initial ADC sfr
-----------------------------------------------------------------*/
void InitADC()
{
    P1ASF = 0xff;                       //Open 8 channels ADC function
    ADC_RES = 0;                        //Clear previous result
    ADC_CONTR = ADC_POWER | ADC_SPEEDLL;
    Delay(2);                           //ADC power-on and delay
}
```

```
/*---------------------------------------------------
Send one byte data to PC
Input: dat (UART data)
Output:-
-----------------------------------------------------------*/
void SendData(BYTE dat)
{
        while (!TI);                    //Wait for the previous data is sent
        TI = 0;                         //Clear TI flag
        SBUF = dat;                     //Send current data
}

/*---------------------------------------------------
Software delay function
-----------------------------------------------------------*/
void Delay(WORD n)
{
        WORD x;
        while (n--)
        {
                x = 5000;
                while (x--);
        }
}
```

Chapter 9 IAP / EEPROM

The ISP in STC15-28-PIN series makes it possible to update the user's application program and non-volatile application data (in IAP-memory) without removing the MCU chip from the actual end product. This useful capability makes a wide range of field-update applications possible. (Note ISP needs the loader program pre-programmed in the ISP-memory.) In general, the user needn't know how ISP operates because STC has provided the standard ISP tool and embedded ISP code in STC shipped samples.But, to develop a good program for ISP function, the user has to understand the architecture of the embedded flash.

The embedded flash consists of 16 pages. Each page contains 512 bytes. Dealing with flash, the user must erase it in page unit before writing (programming) data into it. Erasing flash means setting the content of that flash as FFh. Two erase modes are available in this chip. One is mass mode and the other is page mode. The mass mode gets more performance, but it erases the entire flash. The page mode is something performance less, but it is flexible since it erases flash in page unit. Unlike RAM's real-time operation, to erase flash or to write (program) flash often takes long time so to wait finish.

Furthermore, it is a quite complex timing procedure to erase/program flash. Fortunately, the STC15-28-PIN series carried with convenient mechanism to help the user read/change the flash content. Just filling the target address and data into several SFR, and triggering the built-in ISP automation, the user can easily erase, read, and program the embedded flash.

The In-Application Program feature is designed for user to Read/Write nonvolatile data flash. It may bring great help to store parameters those should be independent of power-up and power-done action. In other words, the user can store data in data flash memory, and after he shutting down the MCU and rebooting the MCU, he can get the original value, which he had stored in.

The user can program the data flash according to the same way as ISP program, so he should get deeper understanding related to SFR IAP_DATA, IAP_ADDRL, IAP_ADDRH, IAP_CMD, IAP_TRIG, and IAP_CONTR.

9.1 IAP / ISP Control Register

The following special function registers are related to the IAP/ISP operation. All these registers can be accessed by software in the user's application program.

Symbol	Description	Address	Bit Address and Symbol (MSB … LSB)	Value after Power-on or Reset
IAP_DATA	ISP/IAP Flash Data Register	C2H		1111 1111B
IAP_ADDRH	ISP/IAP Flash Address High	C3H		0000 0000B
IAP_ADDRL	ISP/IAP Flash Address Low	C4H		0000 0000B
IAP_CMD	ISP/IAP Flash Command Register	C5H	- | - | - | - | - | - | MS1 | MS0	xxxx x000B
IAP_TRIG	ISP/IAP Flash Command Trigger	C6H		xxxx xxxxB
IAP_CONTR	ISP/IAP Control Register	C7H	IAPEN | SWBS | SWRST | CMD_FAIL | - | WT2 | WT1 | WT0	0000 x000B
PCON	Power Control	87H	- | - | LVDF | - | - | - | PD | IDL	xx1x xx00B

IAP_DATA: ISP/IAP Flash Data Register

LSB

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	C2H								

IAP_DATA is the data port register for ISP/IAP operation. The data in IAP_DATA will be written into the desired address in operating ISP/IAP write and it is the data window of readout in operating ISP/IAP read.

IAP_ADDRH: ISP/IAP Flash Address High

LSB

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	C3H								

IAP_ADDRH is the high-byte address port for all ISP/IAP modes.
IAP_ADDRH[7:5] must be cleared to 000, if one bit of IAP_ADDRH[7:5] is set, the IAP/ISP write function must fail.

IAP_ADDRL: ISP/IAP Flash Address Low

LSB

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	C4H								

IAP_ADDRL is the low port for all ISP/IAP modes. In page erase operation, it is ignored.

IAP_CMD: ISP/IAP Flash-operating Mode Command Register

LSB

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	C5H	-	-	-	-	-	-	MS1	MS0

B7~B2: Reserved.
MS1, MS0 : ISP/IAP operating mode selection. IAP_CMD is used to select the flash mode for performing numerous ISP/IAP function or used to access protected SFRs.

0, 0 : Standby
0, 1 : Data Flash/EEPROM read.
1, 0 : Data Flash/EEPROM program.
1, 1 : Data Flash/EEPROM page erase.

IAP_TRIG: ISP/IAP Flash Command Trigger Register.

LSB

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	C6H								

IAP_TRIG is the command port for triggering ISP/IAP activity and protected SFRs access. If IAP_TRIG is filled with sequential 0x5Ah, 0xA5h and if IAPEN(IAP_CONTR.7) = 1, ISP/IAP activity or protected SFRs access will triggered.

IAP_CONTR: ISP/IAP Control Register

bit	Address	B7	B6	B5	B4	B3	B2	B1	B0
name	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	WT2	WT1	WT0

IAPEN : ISP/IAP operation enable.

0 : Global disable all ISP/IAP program/erase/read function.

1 : Enable ISP/IAP program/erase/read function.

SWBS: software boot selection control.

0 : Boot from main-memory after reset.

1 : Boot from ISP memory after reset.

SWRST: software reset trigger control.

0 : No operation

1 : Generate software system reset. It will be cleared by hardware automatically.

CMD_FAIL: Command Fail indication for ISP/IAP operation.

0 : The last ISP/IAP command has finished successfully.

1 : The last ISP/IAP command fails. It could be caused since the access of flash memory was inhibited.

B3: Reserved. Software must write "0" on this bit when IAP_CONTR is written.

WT2~WT0 : Waiting time selection while flash is busy.

Setting wait times			CPU wait times			
WT2	WT1	WT0	Read (2 SYSclks)	Program =55uS	Sector Erase =21mS	Recommended System Clock Frequency (MHz)
1	1	1	2 SYSclks	55 SYSclks	21012 SYSclks	< 1MHz
1	1	0	2 SYSclks	110 SYSclks	42024 SYSclks	< 2MHz
1	0	1	2 SYSclks	165 SYSclks	63036 SYSclks	< 3MHz
1	0	0	2 SYSclks	330 SYSclks	126072 SYSclks	< 6MHz
0	1	1	2 SYSclks	660 SYSclks	252144 SYSclks	< 12MHz
0	1	0	2 SYSclks	1100 SYSclks	420240 SYSclks	< 20MHz
0	0	1	2 SYSclks	1320 SYSclks	504288 SYSclks	< 24MHz
0	0	0	2 SYSclks	1760 SYSclks	672384 SYSclks	< 30MHz

Note: Software reset actions could reset other SFR,but it never influences bits IAPEN and SWBS.The IAPEN and SWBS. The IAPEN and SWBS only will be reset by power-up action, while not software reset.

9.2 IAP/EEPROM Assembly Language Program Introduction

```
;/*It is decided by the assembler/compiler used by users that whether the SFRs addresses are declared by the
DATA or the EQU directive*/
    IAP_DATA    DATA    0C2H    or    IAP_DATA    EQU    0C2H
    IAP_ADDRH   DATA    0C3H    or    IAP_ADDRH   EQU    0C3H
    IAP_ADDRL   DATA    0C4H    or    IAP_ADDRL   EQU    0C4H
    IAP_CMD     DATA    0C5H    or    IAP_CMD     EQU    0C5H
    IAP_TRIG    DATA    0C6H    or    IAP_TRIG    EQU    0C6H
    IAP_CONTR   DATA    0C7H    or    IAP_CONTR   EQU    0C7H

;/*Define ISP/IAP/EEPROM command and wait time*/
    ISP_IAP_BYTE_READ           EQU    1      ;Byte-Read
    ISP_IAP_BYTE_PROGRAM        EQU    2      ;Byte-Program
    ISP_IAP_SECTOR_ERASE        EQU    3      ;Sector-Erase
    WAIT_TIME                   EQU    0      ;Set wait time

;/*Byte-Read*/
    MOV    IAP_ADDRH,    #BYTE_ADDR_HIGH    ;Set ISP/IAP/EEPROM address high
    MOV    IAP_ADDRL,    #BYTE_ADDR_LOW     ;Set ISP/IAP/EEPROM address low
    MOV    IAP_CONTR,    #WAIT_TIME         ;Set wait time
    ORL    IAP_CONTR,    #10000000B         ;Open ISP/IAP function
    MOV    IAP_CMD,      #ISP_IAP_BYTE_READ ;Set ISP/IAP Byte-Read command
    MOV    IAP_TRIG,     #5AH               ;Send trigger command1 (0x5a)
    MOV    IAP_TRIG,     #0A5H              ;Send trigger command2 (0xa5)
    NOP                  ;CPU will hold here until ISP/IAP/EEPROM operation complete
    MOV    A,    IAP_DATA                  ;Read ISP/IAP/EEPROM data

;/*Disable ISP/IAP/EEPROM function, make MCU in a safe state*/
    MOV    IAP_CONTR,    #00000000B         ;Close ISP/IAP/EEPROM function
    MOV    IAP_CMD,      #00000000B         ;Clear ISP/IAP/EEPROM command
    ;MOV   IAP_TRIG,     #00000000B         ;Clear trigger register to prevent mistrigger
    ;MOV   IAP_ADDRH,    #0FFH              ;Move 00 into address high-byte unit,
                                            ;Data ptr point to non-EEPROM area
    ;MOV   IAP_ADDRL,    #0FFH              ;Move 00 into address low-byte unit,
                                            ;prevent misuse

;/*Byte-Program, if the byte is null(0FFH), it can be programmed; else, MCU must operate Sector-Erase firstly,
and then can operate Byte-Program.*/
    MOV    IAP_DATA,     #ONE_DATA          ;Write ISP/IAP/EEPROM data
    MOV    IAP_ADDRH,    #BYTE_ADDR_HIGH    ;Set ISP/IAP/EEPROM address high
    MOV    IAP_ADDRL,    #BYTE_ADDR_LOW     ;Set ISP/IAP/EEPROM address low
    MOV    IAP_CONTR,    #WAIT_TIME         ;Set wait time
    ORL    IAP_CONTR,    #10000000B         ;Open ISP/IAP function
    MOV    IAP_CMD,      #ISP_IAP_BYTE_READ ;Set ISP/IAP Byte-Read command
    MOV    IAP_TRIG,     #5AH               ;Send trigger command1 (0x5a)
    MOV    IAP_TRIG,     #0A5H              ;Send trigger command2 (0xa5)
    NOP                  ;CPU will hold here until ISP/IAP/EEPROM operation complete
```

```
;/*Disable ISP/IAP/EEPROM function, make MCU in a safe state*/
        MOV     IAP_CONTR,      #00000000B      ;Close ISP/IAP/EEPROM function
        MOV     IAP_CMD,        #00000000B      ;Clear ISP/IAP/EEPROM command
        ;MOV    IAP_TRIG,       #00000000B      ;Clear trigger register to prevent mistrigger
        ;MOV    IAP_ADDRH,      #FFH            ;Move 00H into address high-byte unit,
                                                ;Data ptr point to non-EEPROM area
        ;MOV    IAP_ADDRL,      #0FFH           ;Move 00H into address low-byte unit,
                                                ;prevent misuse

;/*Erase one sector area, there is only Sector-Erase instead of Byte-Erase, every sector area account for 512 bytes*/
        MOV     IAP_ADDRH,      #SECTOT_FIRST_BYTE_ADDR_HIGH
                                                ;Set the sector area starting address high
        MOV     IAP_ADDRL,      #SECTOT_FIRST_BYTE_ADDR_LOW
                                                ;Set the sector area starting address low
        MOV     IAP_CONTR,      #WAIT_TIME      ;Set wait time
        ORL     IAP_CONTR,      #10000000B      ;Open ISP/IAP function
        MOV     IAP_CMD,        #ISP_IAP_SECTOR_ERASE   ;Set Sectot-Erase command
        MOV     IAP_TRIG,       #5AH            ;Send trigger command1 (0x5a)
        MOV     IAP_TRIG,       #0A5H           ;Send trigger command2 (0xa5)
        NOP                     ;CPU will hold here until ISP/IAP/EEPROM operation complete

;/*Disable ISP/IAP/EEPROM function, make MCU in a safe state*/
        MOV     IAP_CONTR,      #00000000B      ;Close ISP/IAP/EEPROM function
        MOV     IAP_CMD,        #00000000B      ;Clear ISP/IAP/EEPROM command
        ;MOV    IAP_TRIG,       #00000000B      ;Clear trigger register to prevent mistrigger
        ;MOV    IAP_ADDRH,      #0FFH           ;Move 00H into address high-byte unit,
                                                ; Data ptr point to non-EEPROM area
        ;MOV    IAP_ADDRL,      #0FFH           ;Move 00H into address low-byte unit,
                                                ;prevent misuse
```

9.4 EEPROM Demo Program written in Assembly Language

```
;/*---------------------------------------------------------------------*/
;/* --- STC MCU International Limited ---------------------------------*/
;/* --- STC 1T Series MCU ISP/IAP/EEPROM Demo -------------------------*/
;/* --- Mobile: (86)13922805190 ---------------------------------------*/
;/* --- Fax: 86-755-82944243 ------------------------------------------*/
;/* --- Tel: 86-755-82948412 ------------------------------------------*/
;/* --- Web: www.STCMCU.com -------------------------------------------*/
;/* If you want to use the program or the program referenced in the  */
;/* article, please specify in which data and procedures from STC    */
;/*---------------------------------------------------------------------*/

;/*Declare SFRs associated with the IAP */
IAP_DATA        EQU     0C2H            ;Flash data register
IAP_ADDRH       EQU     0C3H            ;Flash address HIGH
IAP_ADDRL       EQU     0C4H            ;Flash address LOW
IAP_CMD         EQU     0C5H            ;Flash command register
IAP_TRIG        EQU     0C6H            ;Flash command trigger
IAP_CONTR       EQU     0C7H            ;Flash control register

;/*Define ISP/IAP/EEPROM command*/
CMD_IDLE                EQU     0               ;Stand-By
CMD_READ                EQU     1               ;Byte-Read
CMD_PROGRAM             EQU     2               ;Byte-Program
CMD_ERASE               EQU     3               ;Sector-Erase

;/*Define ISP/IAP/EEPROM operation const for IAP_CONTR*/
;ENABLE_IAP     EQU     80H             ;if SYSCLK<30MHz
;ENABLE_IAP     EQU     81H             ;if SYSCLK<24MHz
ENABLE_IAP      EQU     82H             ;if SYSCLK<20MHz
;ENABLE_IAP     EQU     83H             ;if SYSCLK<12MHz
;ENABLE_IAP     EQU     84H             ;if SYSCLK<6MHz
;ENABLE_IAP     EQU     85H             ;if SYSCLK<3MHz
;ENABLE_IAP     EQU     86H             ;if SYSCLK<2MHz
;ENABLE_IAP     EQU     87H             ;if SYSCLK<1MHz

;//Start address for STC12C5A60S2 EEPROM
IAP_ADDRESS EQU 0000H
;------------------------------------------
        ORG     0000H
        LJMP    MAIN
;------------------------------------------
        ORG   0100H
MAIN:
        MOV     P1,     #0FEH           ;1111,1110 System Reset OK
        LCALL   DELAY                   ;Delay
```

```
;-------------------------------
        MOV     DPTR,   #IAP_ADDRESS    ;Set ISP/IAP/EEPROM address
        LCALL   IAP_ERASE               ;Erase current sector
;-------------------------------
        MOV     DPTR,   #IAP_ADDRESS    ;Set ISP/IAP/EEPROM address
        MOV     R0,     #0              ;Set counter (512)
        MOV     R1,     #2
CHECK1:                                 ;Check whether all sector data is FF
        LCALL   IAP_READ                ;Read Flash
        CJNE    A,      #0FFH,  ERROR   ;If error, break
        INC     DPTR                    ;Inc Flash address
        DJNZ    R0,     CHECK1          ;Check next
        DJNZ    R1,     CHECK1          ;Check next
;-------------------------------
        MOV     P1,     #0FCH           ;1111,1100 Erase successful
        LCALL   DELAY                   ;Delay
;-------------------------------
        MOV     DPTR,   #IAP_ADDRESS    ;Set ISP/IAP/EEPROM address
        MOV     R0,     #0              ;Set counter (512)
        MOV     R1,     #2
        MOV     R2,     #0              ;Initial test data
NEXT:                                   ;Program 512 bytes data into data flash
        MOV     A,      R2              ;Ready IAP data
        LCALL   IAP_PROGRAM             ;Program flash
        INC     DPTR                    ;Inc Flash address
        INC     R2                      ;Modify test data
        DJNZ    R0,     NEXT            ;Program next
        DJNZ    R1,     NEXT            ;Program next
;-------------------------------
        MOV     P1,     #0F8H           ;1111,1000 Program successful
        LCALL   DELAY                   ;Delay
;-------------------------------
        MOV     DPTR,   #IAP_ADDRESS    ;Set ISP/IAP/EEPROM address
        MOV     R0,     #0              ;Set counter (512)
        MOV     R1,     #2
        MOV     R2,     #0
CHECK2:                                 ;Verify 512 bytes data
        LCALL   IAP_READ                ;Read Flash
        CJNE    A,  2,  ERROR           ;If error, break
        INC     DPTR                    ;Inc Flash address
        INC     R2                      ;Modify verify data
        DJNZ    R0,     CHECK2          ;Check next
        DJNZ    R1,     CHECK2          ;Check next
;-------------------------------
        MOV     P1,     #0F0H           ;1111,0000 Verify successful
        SJMP    $
;-------------------------------
```

```
ERROR:
        MOV     P0,     R0
        MOV     P2,     R1
        MOV             P3,     R2
        CLR             P1.7                    ;0xxx,xxxx IAP operation fail
        SJMP            $

;/*----------------------------
;Software delay function
;----------------------------*/
DELAY:
        CLR             A
        MOV     R0,     A
        MOV     R1,     A
        MOV     R2,     #20H
DELAY1:
        DJNZ    R0,     DELAY1
        DJNZ    R1,     DELAY1
        DJNZ    R2,     DELAY1
        RET

;/*----------------------------
;Disable ISP/IAP/EEPROM function
;Make MCU in a safe state
;----------------------------*/
IAP_IDLE:
        MOV     IAP_CONTR,      #0              ;Close IAP function
        MOV     IAP_CMD,        #0              ;Clear command to standby
        MOV     IAP_TRIG,       #0              ;Clear trigger register
        MOV     IAP_ADDRH,      #80H            ;Data ptr point to non-EEPROM area
        MOV     IAP_ADDRL,      #0              ;Clear IAP address to prevent misuse
        RET

;/*----------------------------
;Read one byte from ISP/IAP/EEPROM area
;Input: DPTR(ISP/IAP/EEPROM address)
;Output:ACC (Flash data)
;----------------------------*/
IAP_READ:
        MOV     IAP_CONTR,      #ENABLE_IAP     ;Open IAP function, and set wait time
        MOV     IAP_CMD,        #CMD_READ       ;Set ISP/IAP/EEPROM READ command
        MOV     IAP_ADDRL,      DPL             ;Set ISP/IAP/EEPROM address low
        MOV     IAP_ADDRH,      DPH             ;Set ISP/IAP/EEPROM address high
        MOV     IAP_TRIG,       #5AH            ;Send trigger command1 (0x5a)
        MOV     IAP_TRIG,       #0A5H           ;Send trigger command2 (0xa5)
        NOP                     ;MCU will hold here until ISP/IAP/EEPROM operation complete
        MOV     A,      IAP_DATA                ;Read ISP/IAP/EEPROM data
        LCALL   IAP_IDLE                        ;Close ISP/IAP/EEPROM function
        RET
```

```
;/*------------------------------
;Program one byte to ISP/IAP/EEPROM area
;Input: DPAT(ISP/IAP/EEPROM address)
;ACC (ISP/IAP/EEPROM data)
;Output:-
;------------------------------*/
IAP_PROGRAM:
        MOV     IAP_CONTR,      #ENABLE_IAP     ;Open IAP function, and set wait time
        MOV     IAP_CMD,        #CMD_PROGRAM    ;Set ISP/IAP/EEPROM PROGRAM command
        MOV     IAP_ADDRL,      DPL             ;Set ISP/IAP/EEPROM address low
        MOV     IAP_ADDRH,      DPH             ;Set ISP/IAP/EEPROM address high
        MOV     IAP_DATA,       A               ;Write ISP/IAP/EEPROM data
        MOV     IAP_TRIG,       #5AH            ;Send trigger command1 (0x5a)
        MOV     IAP_TRIG,       #0A5H           ;Send trigger command2 (0xa5)
        NOP                     ;MCU will hold here until ISP/IAP/EEPROM operation complete
        LCALL   IAP_IDLE                        ;Close ISP/IAP/EEPROM function
        RET

;/*------------------------------
;Erase one sector area
;Input: DPTR(ISP/IAP/EEPROM address)
;Output:-
;------------------------------*/
IAP_ERASE:
        MOV     IAP_CONTR,      #ENABLE_IAP     ;Open IAP function, and set wait time
        MOV     IAP_CMD,        #CMD_ERASE      ;Set ISP/IAP/EEPROM ERASE command
        MOV     IAP_ADDRL,      DPL             ;Set ISP/IAP/EEPROM address low
        MOV     IAP_ADDRH,      DPH             ;Set ISP/IAP/EEPROM address high
        MOV     IAP_TRIG,       #5AH            ;Send trigger command1 (0x5a)
        MOV     IAP_TRIG,       #0A5H           ;Send trigger command2 (0xa5)
        NOP                     ;MCU will hold here until ISP/IAP/EEPROM operation complete
        LCALL   IAP_IDLE                        ;Close ISP/IAP/EEPROM function
        RET

        END
```

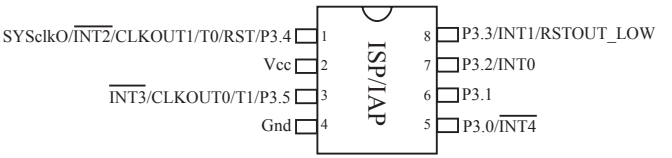
9.5 EEPROM Demo Program written in C Language

```
/*---------------------------------------------------------------------*/
/* --- STC MCU International Limited ---------------------------------*/
/* --- STC 1T Series MCU ISP/IAP/EEPROM Demo -------------------------*/
/* --- Mobile: (86)13922805190 ---------------------------------------*/
/* --- Fax: 86-755-82944243 ------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------*/
/* If you want to use the program or the program referenced in the  */
/* article, please specify in which data and procedures from STC    */
/*---------------------------------------------------------------------*/

#include "reg51.h"
#include "intrins.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

/*Declare SFR associated with the IAP */
sfr  IAP_DATA    =  0xC2;          //Flash data register
sfr  IAP_ADDRH   =  0xC3;          //Flash address HIGH
sfr  IAP_ADDRL   =  0xC4;          //Flash address LOW
sfr  IAP_CMD     =  0xC5;          //Flash command register
sfr  IAP_TRIG    =  0xC6;          //Flash command trigger
sfr  IAP_CONTR   =  0xC7;          //Flash control register

/*Define ISP/IAP/EEPROM command*/
#define   CMD_IDLE        0                 //Stand-By
#define   CMD_READ        1                 //Byte-Read
#define   CMD_PROGRAM 2                     //Byte-Program
#define   CMD_ERASE       3                 //Sector-Erase

/*Define ISP/IAP/EEPROM operation const for IAP_CONTR*/
//#define  ENABLE_IAP      0x80              //if SYSCLK<30MHz
//#define  ENABLE_IAP      0x81              //if SYSCLK<24MHz
#define   ENABLE_IAP      0x82              //if SYSCLK<20MHz
//#define  ENABLE_IAP      0x83              //if SYSCLK<12MHz
//#define  ENABLE_IAP      0x84              //if SYSCLK<6MHz
//#define  ENABLE_IAP      0x85              //if SYSCLK<3MHz
//#define  ENABLE_IAP      0x86              //if SYSCLK<2MHz
//#define  ENABLE_IAP      0x87              //if SYSCLK<1MHz

//Start address for STC12C5A60S2 EEPROM
#define    IAP_ADDRESS    0x0000

void Delay(BYTE n);
void IapIdle();
BYTE IapReadByte(WORD addr);
```

```
void IapProgramByte(WORD addr, BYTE dat);
void IapEraseSector(WORD addr);

void main()
{
	WORD i;

	P1 = 0xfe;                              //1111,1110 System Reset OK
	Delay(10);                              //Delay
	IapEraseSector(IAP_ADDRESS);            //Erase current sector
	for (i=0; i<512; i++)                   //Check whether all sector data is FF
	{
		if (IapReadByte(IAP_ADDRESS+i) != 0xff)
		goto Error;                     //If error, break
	}
	P1 = 0xfc;                              //1111,1100 Erase successful
	Delay(10);                              //Delay
	for (i=0; i<512; i++)                   //Program 512 bytes data into data flash
	{
		IapProgramByte(IAP_ADDRESS+i, (BYTE)i);
	}
	P1 = 0xf8;                              //1111,1000 Program successful
	Delay(10);                              //Delay
	for (i=0; i<512; i++)                   //Verify 512 bytes data
	{
		if (IapReadByte(IAP_ADDRESS+i) != (BYTE)i)
		goto Error;                     //If error, break
	}
	P1 = 0xf0;                              //1111,0000 Verify successful
	while (1);
	Error:
	P1 &= 0x7f;                             //0xxx,xxxx IAP operation fail
	while (1);
}

/*---------------------------
Software delay function
---------------------------*/
void Delay(BYTE n)
{
	WORD x;

	while (n--)
	{
		x = 0;
		while (++x);
	}
}
```

```
/*----------------------------
Disable ISP/IAP/EEPROM function
Make MCU in a safe state
----------------------------*/
void IapIdle()
{
    IAP_CONTR = 0;                  //Close IAP function
    IAP_CMD = 0;                    //Clear command to standby
    IAP_TRIG = 0;                   //Clear trigger register
    IAP_ADDRH = 0x80;               //Data ptr point to non-EEPROM area
    IAP_ADDRL = 0;                  //Clear IAP address to prevent misuse
}

/*----------------------------
Read one byte from ISP/IAP/EEPROM area
Input: addr (ISP/IAP/EEPROM address)
Output:Flash data
----------------------------*/
BYTE IapReadByte(WORD addr)
{
    BYTE dat;                       //Data buffer

    IAP_CONTR = ENABLE_IAP;         //Open IAP function, and set wait time
    IAP_CMD = CMD_READ;             //Set ISP/IAP/EEPROM READ command
    IAP_ADDRL = addr;               //Set ISP/IAP/EEPROM address low
    IAP_ADDRH = addr >> 8;          //Set ISP/IAP/EEPROM address high
    IAP_TRIG = 0x5a;                //Send trigger command1 (0x5a)
    IAP_TRIG = 0xa5;                //Send trigger command2 (0xa5)
    _nop_();                        //MCU will hold here until ISP/IAP/EEPROM
                                    //operation complete
    dat = IAP_DATA;                 //Read ISP/IAP/EEPROM data
    IapIdle();                      //Close ISP/IAP/EEPROM function

    return dat;                     //Return Flash data
}

/*----------------------------
Program one byte to ISP/IAP/EEPROM area
Input: addr (ISP/IAP/EEPROM address)
     dat (ISP/IAP/EEPROM data)
Output:-
----------------------------*/
```

```
void IapProgramByte(WORD addr, BYTE dat)
{
	IAP_CONTR = ENABLE_IAP;		//Open IAP function, and set wait time
	IAP_CMD = CMD_PROGRAM;		//Set ISP/IAP/EEPROM PROGRAM command
	IAP_ADDRL = addr;		//Set ISP/IAP/EEPROM address low
	IAP_ADDRH = addr >> 8;		//Set ISP/IAP/EEPROM address high
	IAP_DATA = dat;			//Write ISP/IAP/EEPROM data
	IAP_TRIG = 0x5a;		//Send trigger command1 (0x5a)
	IAP_TRIG = 0xa5;		//Send trigger command2 (0xa5)
	_nop_();			//MCU will hold here until ISP/IAP/EEPROM
					//operation complete
	IapIdle();
}

/*----------------------------
Erase one sector area
Input: addr (ISP/IAP/EEPROM address)
Output:-
-----------------------------*/
void IapEraseSector(WORD addr)
{
	IAP_CONTR = ENABLE_IAP;		//Open IAP function, and set wait time
	IAP_CMD = CMD_ERASE;		//Set ISP/IAP/EEPROM ERASE command
	IAP_ADDRL = addr;		//Set ISP/IAP/EEPROM address low
	IAP_ADDRH = addr >> 8;		//Set ISP/IAP/EEPROM address high
	IAP_TRIG = 0x5a;		//Send trigger command1 (0x5a)
	IAP_TRIG = 0xa5;		//Send trigger command2 (0xa5)
	_nop_();			//MCU will hold here until ISP/IAP/EEPROM
					//operation complete
	IapIdle();
}
```



SOP-8/DIP-8